GROUP ASSIGNMENT

CT124-3-3-BCD

BLOCKCHAIN DEVELOPMENT

HAND OUT DATE : Week 5

HAND IN DATE : 22-12-2023

WEIGHTAGE : 60%

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the Moodle System.**
2. **Students are advised to underpin their answers with the use of references (cited using the APA Style System of Referencing)**
3. **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**
4. **Cases of plagiarism will be penalized.**
5. **The assignment should be submitted in softcopy, where the softcopy of the written assignment and source code (where appropriate) should be on Moodle System.**
6. **You must obtain 50% overall to pass this module.**

No.	Name	TP No.	Intake Code
1	Yam Chen Xi	TP061635	APD3F2308CS
2	Soh Myles	TP061320	APD3F2308CS

Table of Contents

1.0 Introduction

The implementation of the land registration system is based on the seamless integration of blockchain technology and cryptographic methods, ensuring a secure, transparent, and easily accessible framework for essential data. This report thoroughly examines the systematic deployment of key elements, each customized to enhance the overall system. Core sections delve into the intricacies of blockchain technology, hashing algorithms, cryptographic approaches, digital signatures, and immutability techniques. A comprehensive exploration of these elements elucidates their specific implementations and collective contributions to strengthening the system's robustness. In addition to theoretical discussions, the report provides a practical understanding of the system's operations through a visual walkthrough accompanied by screenshots. This immersive exploration offers stakeholders first-hand experience with user interfaces and functionalities that foster a tangible grasp of the system’s capabilities. The subsequent evaluation section rigorously assesses the performance across critical dimensions such as data security, transparency, and accessibility. This analytical perspective delivers valuable insights into the effectiveness of integrated technologies, presenting a holistic view of both strength and potential areas for improvement.

2.0 Solution Implementation

2.1 Blockchain

Blockchain is a digital ledger that uses consensus protocols and cryptography to function in a decentralized and secure manner. It is categorized as distributed ledger technology (DLT) and uses an open network's shared database to process, record, and authenticate transactions. It is essentially a decentralized ledger that keeps track of who owns what and how it is transferred across a distributed network of computers, and it is available to all users. Apart from its fundamental function in cryptocurrencies, blockchain functions as an adaptable platform that accommodates an extensive array of applications, encompassing smart contracts, distributed ledgers customized for diverse sectors and objectives, and cryptocurrencies. (IBM, n.d.)

A blockchain is made up of data blocks connected by cryptographic links. A block cannot be removed or changed once it is added to the chain; each block records multiple transactions. Because of this, blockchains are extremely safe and impervious to manipulation. In the context of blockchain, a block is a grouping of data that is combined with other blocks to create a chain. Every block in the chain is uniquely identified by a code known as a hash, which also connects it to the block before it. In doing so, an unbreakable chain of blocks is created that can be used to securely and store and verify decentralized data. (Sajana et al., 2018)

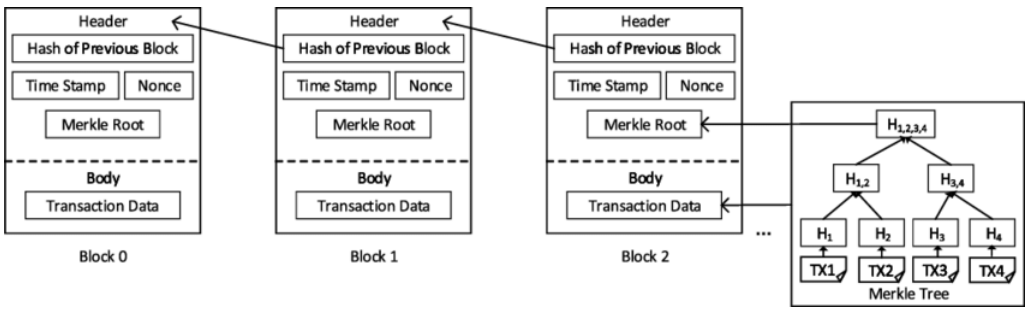

Figure 1: Structure of Blockchain

2.1.1 Implementation of Blockchain

Block.java

```
1  package blockchain;
2
3  import java.io.Serializable;
5
6  public class Block implements Serializable{
7      //create blocks here
8      /* relationship implementation */
9      public Header header;
10     public TransactionCollection tranxLst;
11
12     public Block(String previousHash, String root) {
13         /* composition relationship */
14         this.header = new Header();
15         header.setTimestamp( new Timestamp(System.currentTimeMillis()).getTime() );
16         header.setPreviousHash(previousHash);
17         String info = String.join("+",
18                 Integer.toString(header.getIndex()),
19                 header.getPreviousHash(),
20                 root,
21                 Long.toString(header.getTimestamp())
22                 );
23         header.currentHash = Hasher.sha256ns(info);
24     }
25
26
27     public TransactionCollection getTransactions()
28     {
29         return this.tranxLst;
30     }
31
32     /* aggregation relationship */
33     public void setTransactions( TransactionCollection tranxLst )
34     {
35         this.tranxLst = tranxLst;
36     }
37
```

Block.java

```
38     public Header getHeader()
39     {
40         return this.header;
41     }
42
43     @Override
44     public String toString() {
45         return "Block [header=" + header + ", tranxLst=" + tranxLst + "]";
46     }
```

Block.java

```
47
48      /* define inner class for the Header */
49      public class Header implements Serializable{
50          public int index;
51          public String currentHash, previousHash;
52          public long timestamp;
53
54          @Override
55          public String toString() {
56              return "Header [index=" + index + ", currentHash=" + currentHash +
57                      ", previousHash=" + previousHash + ", timestamp=" + timestamp + "]";
58          }
59
60          //getset methods
61          public int getIndex() {
62              return index;
63          }
64
65          public void setIndex(int index) {
66              this.index = index;
67          }
68
69          public String getCurrentHash() {
70              return currentHash;
71          }
72
73          public void setCurrentHash(String currentHash) {
74              this.currentHash = currentHash;
75          }
76
77          public String getPreviousHash() {
78              return previousHash;
79          }
80
81          public void setPreviousHash(String previousHash) {
82              this.previousHash = previousHash;
83          }
84
85          public long getTimestamp() {
86              return timestamp;
87          }
88
89          public void setTimestamp (long timestamp) {
90              this.timestamp = timestamp;
91          }
92
93      }
94  }
```

Figure 2 & 3 & 4 & 5: Block.java code snippet

The following figures as shown above is the block class that compiles into serializable interface. All of the necessary components to create the block are all shown above, these include:

1. Header of the block
2. The current hash of the block
3. The previous hash of the block before
4. The block's time stamp in Epoch time

2.2 Hashing

A hashing algorithm is a mathematical operation that accepts an input, also known as a "message," and outputs a fixed-length character string, typically consisting of a series of letters and numbers. A digest, also known as a hash value, is the result of a hash function. Numerous branches of computer science make use of hash functions. An input message of any length [1] can be processed by the MD (Message Digest) family of cryptographic hash functions to produce a fixed-size output (128, 160, 256, or 512 bits). SHA-1 and MD5 are the two most used MD algorithms. Unfortunately, SHA-1 is also being phased out with a fixed-size output (224, 256, 384, or 512 bits) from an input message of any length, and MD5 is no longer regarded as secure because of its vulnerabilities. SHA-256 and SHA-512 are the two SHA algorithms that are most frequently used. Digital signatures, message authentication codes, and other security applications make extensive use of these algorithms. The newest member of the SHA family, SHA-3, was created to take the place of SHA-2. (Krishnapriya & Sarath, 2020)

2.2.1 Implementation of Hashing Algorithm

Hasher.java

```
15
16      //with salt
17      public static String sha256(String input, byte[] salt) {
18          return hash(input,salt,"SHA-256");
19      }
20
21      //without salt
22      public static String sha256ns(String input)
23      {
24          return hashns( input, "SHA-256" );
25      }
26
```

```
27      //with salting
28      public static String hash(String input, byte[] salt, String algorithm)
29      {
30      String hashCode = "";
31      try {
32          MessageDigest md = MessageDigest.getInstance(algorithm);
33          md.update(input.getBytes() );
34          md.update(salt);
35          byte[] hashBytes = md.digest();
36          hashCode = Base64.getEncoder().encodeToString(hashBytes);
37      } catch (Exception e) {
38      e.printStackTrace();
39      }
40      return hashCode;
41      }
42
43      //without salting
44      private static String hashns(String input, String algorithm)
45      {
46      String hashCode = "";
47      try {
48          MessageDigest md = MessageDigest.getInstance(algorithm);
49          md.update( input.getBytes() );
50          //digesting...
51          byte[] hashBytes = md.digest();
52          //convert the byte[] to String
53          //1)
54          hashCode = Base64.getEncoder().encodeToString(hashBytes);
55          //2) hex format output - recommended!
56          //hashCode = Hex.encodeHexString(hashBytes);
57      } catch (Exception e) {
58      e.printStackTrace();
59      }
60      return hashCode;
61      }
62
63  }
```

Figure 6 & 7: Hasher.java code snippet

The figures above shows the Hashing Algorithm that is used for the system. SHA-256 is used for the hashing algorithm and both salted and non-salted hashing is utilized in the system.

2.2.2 Implementation of Salt Algorithm

```
 8
 9      public static byte[] generate() {
10          SecureRandom sr = new SecureRandom();
11          byte[] b = new byte[64];
12          sr.nextBytes(b);
13          return b;
14      }
15
```

Figure 8: Salt code snippet

This figure shows the generation of the salt for the hashing with salt function.

2.3 Cryptographic Algorithm

Cryptography is the process of protecting correspondence from hostile parties 1. It involves converting plaintext into ciphertext—which is only readable by authorized parties with the decryption key—using mathematical algorithms. Secrecy is one of the characteristics of cryptography; only the intended recipient can access data; others cannot. Integrity: Information cannot be added to or changed while it is being stored between the sender and the recipient. Non-repudiation: The person who created or sent the data cannot retract their plan to send it later. Authentication: The sender and recipient's identities are verified.

Symmetric encryption is a kind of encryption where the encryption and decryption processes share the same key. While less secure than public-key cryptography, it is faster and more effective. Applications like data storage and transmission, where efficiency and speed are more crucial than security, use symmetric encryption. Asymmetric cryptography, commonly referred to as public-key cryptography, employs a pair of keys: a public key and a private key. Data is encrypted using the public key and decrypted using the private key. While more secure than symmetric encryption, public-key cryptography operates more slowly. It is utilized in applications like digital signatures and secure communication protocols where security is more crucial than speed.

2.3.1 Implementation of Symmetric Encryption

PredefinedCharsSecretKey.java

```java
package blockchain;

import java.security.Key;

public class PredefinedCharsSecretKey {
    private static final String ALGORITHM = "AES";
    private static final String SECRET_CHARS = "asddsaqweewqpbnmzxccz098123765543";

    public static Key create()
    {
        int keySize = 16;
        return new SecretKeySpec (Arrays.copyOf(SECRET_CHARS.getBytes(), keySize), ALGORITHM);

    }
}
```

Figure 9: PredefinedCharsSecretKey.java code snippet

RandomSecretKey.java

```java
package blockchain;

import java.security.Key;

public class RandomSecretKey {
    private static final String ALGORITHM = "AES";

    public static Key create()
    {
        short keySize = 256;
        try {
            KeyGenerator kg = KeyGenerator.getInstance(ALGORITHM);
            kg.init(keySize, new SecureRandom());
            return kg.generateKey();

        }catch (Exception e) {
            e.printStackTrace();
            return null;
        }

    }
}
```

Figure 10: RandomSecretKey.java code snippet

Symmetric.java

```java
package blockchain;

import java.security.Key;

public class Symmetric {
    static Cipher cipher;
    Symmetric(String algorithm) throws Exception{
        cipher = Cipher.getInstance(algorithm);
    }
    public Symmetric()throws Exception{
        this("AES");
    }
    public static String encrypt(String data, Key key) throws Exception
    {
        String cipherText = null;
        cipher.init(Cipher.ENCRYPT_MODE, key);

        byte[] cipherBytes = cipher.doFinal(data.getBytes());
        cipherText = Base64.getEncoder().encodeToString(cipherBytes);
        return cipherText;
    }
    public String decrypt(String cipherText, Key key)throws Exception
    {
        cipher.init(Cipher.DECRYPT_MODE,key);
        byte[] cipherBytes = Base64.getDecoder().decode(cipherText);
        byte[] dataBytes = cipher.doFinal(cipherBytes);
        return new String (dataBytes);
    }
}
```

Figure 11: Symmetric.java code snippet

The figures above show the symmetric function to encrypt the data for security and the protection of the data. Theres a class that is called PredefinedCharsSecretKey in which produces the secret key. Since there are multiple functions that are going to be interacting with the land registry like, the transfer of ownership, the transaction of money that is for registering or purchasing a land etc. The data that is passed into the system is then encrypted using the secret key.

2.4 Digital Signature

Digital signatures play a crucial role in blockchain technology, ensuring the authenticity and integrity of digital messages or documents. This cryptographic technique involves encrypting a hash of the message or document with a private key, which can then be decrypted using the corresponding public key to verify that no tampering has occurred. In blockchains, digital signatures are primarily utilized for transaction authentication and authorization validation to maintain trust and security (coinbase, 2022).

2.4.1 Implementation of Digital Signature

```
1  package blockchain;
2
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8  import java.security.*;
9
10 public class DigitalSignature {
11     private static final String ALGORITHM = "SHA256WithRSA";
12     private static final String KEY_PAIR_FILE = "keypair.dat";
13
14     private static DigitalSignature instance;
15     private Signature signature;
16     private KeyPair keyPair;
17
18     private DigitalSignature() {
19         // Load existing key pair or generate a new one
20         keyPair = loadKeyPair(KEY_PAIR_FILE);
21         if (keyPair == null) {
22             keyPair = generateKeyPair();
23             saveKeyPair(KEY_PAIR_FILE, keyPair);
24         }
25     }
26
27     // Singleton pattern: Get the instance of DigitalSignature
28     public static DigitalSignature getInstance() {
29         if (instance == null) {
30             instance = new DigitalSignature();
31         }
32         return instance;
33     }
34
```

Figure 12: Initialization and Singleton Pattern of Digital Signature

The code snippet shows the DigitalSignature class, which is responsible for handling cryptographic key pairs and executing digital signature functions in a blockchain application. This class adheres to the Singleton pattern to guarantee a solitary instance exists across the application. It utilizes the SHA256 hashing algorithm along with RSA for creating and authenticating digital signatures.

```
35      // Generate key pair for digital signature
36      private KeyPair generateKeyPair() {
37          try {
38              KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
39              keyPairGenerator.initialize(2048);
40              return keyPairGenerator.generateKeyPair();
41          } catch (NoSuchAlgorithmException e) {
42              e.printStackTrace();
43              throw new RuntimeException("Error generating key pair.");
44          }
45      }
46
47      // Load key pair from file
48      private KeyPair loadKeyPair(String fileName) {
49          try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
50              return (KeyPair) ois.readObject();
51          } catch (IOException | ClassNotFoundException e) {
52              return null; // File not found or error loading, return null
53          }
54      }
55
56      // Save key pair to file
57      private void saveKeyPair(String fileName, KeyPair keyPair) {
58          try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {
59              oos.writeObject(keyPair);
60          } catch (IOException e) {
61              e.printStackTrace();
62              throw new RuntimeException("Error saving key pair.");
63          }
64      }
65
66      // Getter for public key
67      public PublicKey getPublicKey() {
68          return keyPair.getPublic();
69      }
70
71      // Setter for key pair
72      public void setKeyPair(KeyPair keyPair) {
73          this.keyPair = keyPair;
74      }
```

Figure 13: Key Pair Management

The DigitalSignature class incorporates code for managing key pairs, including their generation, loading, and saving for digital signature purposes. The generateKeyPair method initializes a KeyPairGenerator with the RSA algorithm and a key size of 2048 bits to create a new key pair. Using object serialization, the loadKeyPair method retrieves a previously saved key pair from a specified file; in case of failure or absence of the file, it returns null. Similarly, utilizing object serialization as well, the saveKeyPair method permanently stores an inputted key pair into a file. Through the getPublicKey method one can gain access to the public key of the current generated pair while using setKeyPair facilitates manual assignment of retrieved or externally provided keys via deserialization. These methods collectively ensure secure and efficient management of RSA key pairs utilized in digital signature operations.

```
76      // Generate digital signature for the transaction
77      public byte[] generateDigitalSignature(String data) throws NoSuchAlgorithmException,
78      InvalidKeyException, SignatureException {
79          signature = Signature.getInstance(ALGORITHM);
80          signature.initSign(keyPair.getPrivate());
81          signature.update(data.getBytes());
82          return signature.sign();
83      }
84
85      // Verify digital signature during transaction approval
86      public boolean verifyDigitalSignature(String data, byte[] signatureBytes)
87              throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
88          Signature signature = Signature.getInstance(ALGORITHM);
89          signature.initVerify(keyPair.getPublic());
90          signature.update(data.getBytes());
91          return signature.verify(signatureBytes);
92      }
```

Figure 14: Digital Signature Generation and Verification

The code snippet provided includes functions for creating and authenticating digital signatures using the RSA algorithm. The generateDigitalSignature method initializes a Signature instance with the SHA-256 with RSA algorithm, configures it to sign mode using the private key from the current key pair, updates the signature with the byte representation of input data, and then returns the generated digital signature as an array of bytes. Meanwhile, verifyDigitalSignature method sets up a new Signature instance in verify mode with identical algorithm settings. It verifies against public key from current key pair, updates signature with byte representation of input data, and finally indicates whether provided signature bytes match computed signature via boolean return value. These functions collectively facilitate secure generation and validation of digital signatures for transactions within blockchain applications while ensuring both integrity and authenticity of data.

```
201             while (attempts < MAX_ATTEMPTS) {
202                 System.out.print("\nEnter your password for validation ('X' to cancel): ");
203                 enteredPassword = scanner.nextLine();  // Use nextLine() to consume the entire line
204 
205                 if (enteredPassword.equalsIgnoreCase("X")) {
206                     System.out.println("\n** Transaction canceled. **");
207                     return false;
208                 }
209 
210                 if (enteredPassword.equals(login.getCurrentUserPass())) {
211                     System.out.println("** Transaction successful. **");
212 
213                     int transID = generateNewTransRecID(trans);
214                     Timestamp recDate = new Timestamp(System.currentTimeMillis());
215 
216                     status tranStatus = enuum.status.PENDING;
217 
218                     String transactionData = transID + "_" + landID + "_" + buyerID + "_" + sellerID;
219 
220                     myDigitalSignature = digitalSignature.generateDigitalSignature(transactionData);
221 
222                     // Create a new transaction object
223                     TransRec newTransaction = new TransRec(transID, landID, buyerID, sellerID, recDate,
224                             amount, paymentMethod, transType, tranStatus, myDigitalSignature);
225 
226                     String transactionString = newTransaction.toString();
227                     Blockchain.createBlockchain(transactionString);
228 
229                     FileHandler.addObject(newTransaction, TRANSACTION_FILE);
230                     displayCurrentTransactionInfo(transID);
231 
232                     lrh.newLandRec(mode, landID, buyerID, transID);
233 
234                     return true;
235                 } else {
236                     attempts++;
237                     System.out.println("Invalid password. Attempts remaining: " + (MAX_ATTEMPTS - attempts));
238                 }
```

Figure 15: Usage of Digital Signature Generation in newTransaction Method

The newTransaction function commences the process of initiating a new transaction. This includes creating a digital signature to safeguard the transaction data before incorporating it into the blockchain. Upon user initiation of a transaction, the method computes the transaction amount based on the selected mode, requests for payment method information, and authenticates the user's password. Once authenticated, a unique transaction ID is produced, and using the generateDigitalSignature method from the DigitalSignature class to sign relevant details associated with this transmission. The resulting digital signature becomes linked with that specific transmission; subsequently added along with it onto the blockchain.This procedure ensures that such transactions remain impervious to tampering and can be subsequently verified during an approval process.

```
278             if (transactionToApprove != null) {
279                 String transactionData = transactionToApprove.getTransID() + "_"
280                         + transactionToApprove.getLandID() + "_"
281                         + transactionToApprove.getBuyerID() + "_"
282                         + transactionToApprove.getSellerID();
283 
284                 if (digitalSignature.verifyDigitalSignature(transactionData, transactionToApprove.getMyDigitalSignature())) {
285                     transactionToApprove.setTranStatus(enuum.status.COMPLETE);
286 
287                     System.out.println("** Transaction ID #" + transIDToApprove + " approved. **");
288                     String transactionString = transactionToApprove.toString();
289                     Blockchain.createBlockchain(transactionString);
290                 } else {
291                     System.out.println("** Digital signature verification failed for Transaction ID #" + transIDToApprove + ". **");
292                 }
293             } else {
294                 System.out.println("** Transaction not found or already approved. **");
295             }
```

Figure 16: Usage of Digital Signature Verification in approveTransaction Method

The approveTransaction function is responsible for approving pending transactions. It first displays a list of transactions with a pending status and prompts the user to input the transaction ID to approve. Upon user input, the method retrieves the corresponding transaction from the list and verifies its digital signature using the verifyDigitalSignature method. If the verification is successful, indicating that the transaction data has not been altered and was signed by the rightful owner, the transaction status is updated to COMPLETE. Subsequently, a new blockchain entry is created for the approved transaction. This use of digital signatures ensures that only authorized entities can approve transactions and guarantees the authenticity of the transactions within the blockchain.

2.5 Immutability Technique

Data immutability is the concept of retaining the data to exactly what was written and cannot be changed or altered in any way. Databases have their data mutable, meaning that the data can be read, updated or deleted. However, data immutability only allows the data to be read-only, and any new changes has to be appended as new data instead of modifying the pre-existing data. There are numerous benefits of having immutable data for land registration. Data integrity ensures that the original information and data remain intact and cannot be altered. This is crucial for maintaining the trustworthiness of the data and the integrity of it. Next is the reproducibility. Immutable data allows an easy reproduction of past analysis and calculations since the data is always constant. This allows a more accurate representation for data-driven decisions and maintaining consistency. Data security is definitely the most important when it comes to unaltered data. It adds an additional layer of security as it prevents unwanted and unauthorized modifications or tampering. (Dremio, n.d.) There are multiple ways to achieve data immutability, and one of the ways is Merkle Tree in which blockchain technology requires to make sure data is always consistent as it serves as a digital ledger.

2.5.1 Implementation of Merkle Tree

A Merkle tree, also known as a hash tree, is a data structure used for data verification and synchronization. It is a tree data structure where each non-leaf node is a hash of its child nodes. All the leaf nodes are at the same depth and are as far left as possible. It maintains data integrity and uses hash functions for this purpose. The root hash is used as the fingerprint for the entire data. This structure of the tree allows efficient mapping of huge data, and small changes made to the data can be easily identified. If we want to know where data change has occurred, then we can check if data is consistent with the root hash, and we will not have to traverse the whole structure but only a small part of the structure.

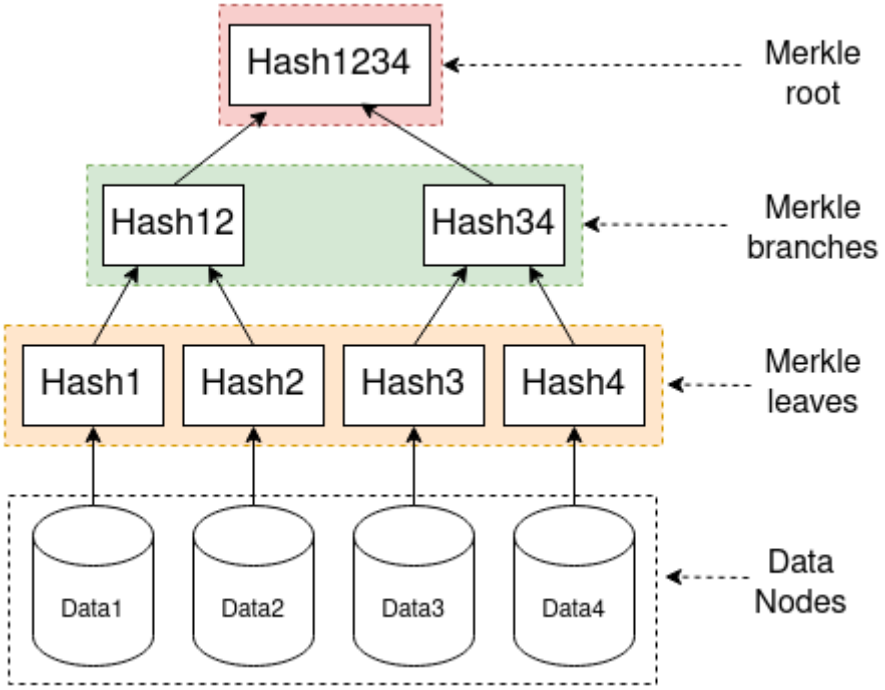

Figure 17: Merkle Tree Struture

MerkleTree.java

```java
package blockchain;

import java.util.ArrayList;

public class MerkleTree {
    private List<String> tranxLst;
    public String root = "0" ;

    public String getRoot() {
        return root;
    }

    /**
     * @implNote
     * Set the transaction list to the MerkleTree object.
     *
     * @param tranxLst
     */
    private MerkleTree(List<String> tranxLst) {
        super();
        this.tranxLst = tranxLst;

    }

    private static MerkleTree instance;
    public static MerkleTree getInstance( List<String> tranxLst ) {
        if( instance == null ) {
            return new MerkleTree(tranxLst);
        }
        return instance;
    }

    /**
     * @implNote
     * Build merkle tree
     *
     * @implSpec
     * + build() : void
     */
```

MerkleTree.java

```
     * @implSpec
     * + build() : void
     */
    public void build() {

        List<String> tempLst = new ArrayList<>();

        for (String tranx : this.tranxLst) {
            tempLst.add(tranx);
        }

        List<String> hashes = genTranxHashLst( tempLst );
        while(  hashes.size() != 1 ) {
            hashes = genTranxHashLst( hashes );
        }
        this.root = hashes.get(0);
    }

    /**
     * @implNote
     * Generate hashes of transactions
     *
     * @implSpec
     * - genTranxHashLst(List<String>) : List<String>
     */
    private List<String> genTranxHashLst(List<String> tranxLst) {
        List<String> hashLst = new ArrayList<>();
        int i = 0;
        while( i < tranxLst.size() ) {

            String left = tranxLst.get(i);
            i++;

            String right = "";
            if( i != tranxLst.size() ) right = tranxLst.get(i);

            String hash = Hasher.sha256(left.concat(right), null);
            hashLst.add(hash);
            i++;
        }
        return hashLst;
    }
}
```

Figure 18 & 19: MerkleTree.java code snippet

The figures here shown the process in the creation of the Merkle tree. The code will try to retrieve previous transactions that has been made and store them into an array list, this array list is then hashed for transaction that has been made previously until a singular hash which is the Merkle Root.

2.5.2 Implementation of Shared Ledger

Figure 20: chain.bin output

The figure above shows the shared ledger for this system, it consists of the header of each block, the current hash, the previous hash, timestamp, and the transaction collection list which is encrypted for better security.

Blockchain.java

```
1  package blockchain;
2
3  import java.io.*;
4  import java.security.Key;
5  import java.util.Base64;
6  import java.util.LinkedList;
7  import javax.crypto.Cipher;
8
9  import com.google.gson.GsonBuilder;
10
11 public class Blockchain {
12     private static LinkedList<Block>db = new LinkedList<>();
13     private static Cipher cipher;
14     private static Blockchain _instance;
15     public static Blockchain getInstance(String chainFile) {
16         if (_instance == null)
17             _instance = new Blockchain(chainFile);
18         return _instance;
19     }
20     public String chainFile;
21     public Blockchain(String chainFile) {
22         super();
23         this.chainFile = chainFile;
24         try {
25             // Create a Cipher object with the AES algorithm
26             cipher = Cipher.getInstance("AES");
27             System.out.println("> Blockchain object is created!");
28         } catch (Exception e) {
29             e.printStackTrace();
30         }
31     }
32
33     public void genesis() {
34         Block genesis = new Block ("0", "Root");
35         db.add(genesis);
36         persist();
37     }
38
39     public void nextBLock(Block newBlock)
40     {
41         db = get();
42         db.add(newBlock);
43         persist();
44     }
```

Blockchain.java

```
45
46      public LinkedList<Block> get(){
47          try(FileInputStream fin = new FileInputStream(this.chainFile);
48                  ObjectInputStream in = new ObjectInputStream(fin);
49                  ){
50              return (LinkedList<Block>)in.readObject();
51          }catch(Exception e) {
52              e.printStackTrace();
53              return null;
54          }
55      }
56
57      public void persist()
58      {
59          try(FileOutputStream fout = new FileOutputStream(this.chainFile);
60                  ObjectOutputStream out = new ObjectOutputStream(fout);){
61              out.writeObject(db);
62              System.out.println(">> Master file is updated!");
63          }catch(Exception e) {
64              e.printStackTrace();
65          }
66      }
67
68      public void persistTextFile() {
69          try (FileWriter writer = new FileWriter(this.chainFile + ".txt")) {
70              String chain = new GsonBuilder().setPrettyPrinting().create().toJson(db);
71              writer.write(chain);
72              System.out.println(">> Text file is updated!");
73          } catch (Exception e) {
74              e.printStackTrace();
75          }
76      }
77      private static String masterFolder = "master";
78      private static String fileName=masterFolder+"/chain.bin";
79
```

```
80      public static void createBlockchain(String data) {
81      Key secretKey = PredefinedCharsSecretKey.create();
82      Blockchain bc = Blockchain.getInstance(fileName);
83      String encryptedChain = null;
84      try {
85          encryptedChain = encrypt(data, secretKey);
86      } catch (Exception e) {
87          // TODO Auto-generated catch block
88          e.printStackTrace();
89      }
90      if (!new File(masterFolder).exists()) {
91          System.err.println("> creating Blockchain binary!");
92          new File(masterFolder).mkdir();
93          bc.genesis();
94          }
95      else {
96          String line1=encryptedChain;
97          TransactionCollection tranxLst = new TransactionCollection();
98          tranxLst.add(line1);
99          String previousHash = bc.get().getLast().getHeader().getCurrentHash();
100         Block b1 = new Block(previousHash, "Root");
101         b1.setTransactions(tranxLst);
102         bc.nextBLock(b1);
103         bc.persistTextFile();
104     }
105 }
106 }
107
```

Figure 21 & 22 & 23: Blockchain.java code snippet

The figures as shown above are the required processes in creating the blockchain and storing the salt-hashed data that is also encrypted into a .bin for the distribution of the ledger itself.

The data that is encrypted is stored in a string format which can be decrypted.

TransactionCollection.java

```java
package blockchain;

import java.io.Serializable;

public class TransactionCollection implements Serializable {
    private final int SIZE = 10;

    public String merkleRoot;

    public List<String> tranxLst;

    public void complete() {
        MerkleTree mt = MerkleTree.getInstance(tranxLst);
        mt.build();
        this.merkleRoot = mt.getRoot();

    }

    public TransactionCollection() {
        tranxLst = new ArrayList<>(SIZE);
        }

    public void add(String tranx) {
        tranxLst.add(tranx);
        }

    public String getMerkleRoot() {
        return this.merkleRoot;
        }

    public List<String> getTransactionList(){
        return this.tranxLst;
    }
}
```

Figure 24: TransactionCollection.java code snippet

To set the transaction list, the transaction collection class provides methods like set and getMerkleRoot(). Furthermore, transactions are repeatedly hashed to create the Merkle Trees. In order to pair transactions, a temporary list is created first, followed by a list of hashes, and so on, until the final hash is created, which serves as the Merkle Tree's root.

3.0 System Screenshot

3.1 Main Menu

```
Land Registration System
-------------------------------------------------
1. Login
2. Register
3. Exit
Enter your choice: 1
```

Figure 25: Screenshot of Main Menu CLI

The users are welcome to the Land Registration System with the main menu consisting of two features which are login and register.

3.1.1 Login

```
Login
-------------------------------------------------
Username: admin
Password: admin123
** Login successful! **
```

Figure 26: Screenshot of Login CLI

To login to the system, user is required to input username and password that had been registered into the system. There is only one admin in the system, which using username “admin” and password “admin123”.

3.1.2 Register User

```
Register User
-------------------------------------------------
User ID #5
Username        : sabrina
Password        : sabrina123
Age             : 25
Email           : sabrina@gmail.com
Phone Number    : 0111020930
Occupation      : Singer
** New user registered successfully. **
** You can now login with your credentials. **
```

Figure 27: Screenshot of Register CLI

Register user function is used by first time user of the system, where they can register their credentials and user information into the system.

3.2 Admin

3.2.1 Menu

```
Welcome, admin!

Admin Menu
-----------------------------------------------------
1. Manage User
2. Manage Land
3. Manage Transaction
4. Land Record
5. Logout
Enter your choice:
```

Figure 28: Screenshot of Admin Menu CLI

Admin menu is the landing page upon login successfully as administrator. Admin is granted the authorization to manage user, land, transaction and land record of all customers.

3.2.2 Manage User

```
Manage User
-----------------------------------------------------
1. User List
2. Register New User
3. Modify User
4. Delete User
5. Back
Enter your choice:
```

Figure 29: Screenshot of Manage User CLI

In manage user, admin can access to the user list, register new user, modify user information and delete user.

```
                                                    User
-------------------------------------------------------------------------------------------------------------------------------
ID   | User Type  | Username        | Password        | Age | Email                | Phone Number    | Occupation
-------------------------------------------------------------------------------------------------------------------------------
1    | ADMIN      | admin           | admin123        | 0   | NULL                 | NULL            | NULL
2    | CUSTOMER   | chenxi          | chenxi123       | 21  | chenxi@gmail.com     | 0111002836      | Artist
3    | CUSTOMER   | myles           | myles456        | 21  | myles@gmail.com      | 01110092837     | Doctor
4    | CUSTOMER   | joshua          | joshua123       | 25  | joshua@gmail.com     | 0192836192      | Lawyer
5    | CUSTOMER   | sabrina         | sabrina123      | 25  | sabrina@gmail.com    | 0111020930      | Singer
-------------------------------------------------------------------------------------------------------------------------------
```

Figure 30: Screenshot of User List CLI

Admin is able to get all the user information through the user list.

```
User ID #5
Username        : sabrina
Password        : sabrina123
Age             : 25
Email           : sabrina@gmail.com
Phone Number    : 0111020930
Occupation      : Singer
** New user registered successfully. **
```

Figure 31: Screenshot of Register New User CLI

Admin is able to register new user with the auto generated user ID and input the required information for new user.

```
Enter the username or user ID to modify: sabrina
New username            : carpenter
New password            : carpenter123
New age                 : 25
New email               : carpenter@gmail.com
New phone number        : 0111020930
New occupation          : Singer


** User modified successfully. **
```

Figure 32: Screenshot of Modify User CLI

To modify user, admin is required to input username or user ID to access to the selected user's original information and input new information to update the user information.

```
Enter the username or user ID to delete: 5
** User deleted successfully. **
```

Figure 33: Screenshot of Delete User CLI

To delete user, admin is required to input username or user ID to delete the selected user from the system.

3.2.3 Manage Land

```
Manage Land
-----------------------------------------------------
1. Land List
2. Register New Land
3. Approve Land Registration
4. Back
Enter your choice:
```

Figure 34: Screenshot of Delete User CLI

In manage land, admin is granted permission to access to all user's land information, register new land and approve land registration.

Registered Land

ID	Area	Height	Volume	Year	Owner	Registered Date	Condition	Value	Status
1	4000.0	300.0	120000.0	2020	3	2023-12-20 23:46:07.281	4	RM800000.00	COMPLETE
2	3000.0	500.0	1500000.0	2023	3	2023-12-20 23:46:41.773	5	RM2000000.00	COMPLETE
3	3000.0	300.0	900000.0	2019	4	2023-12-20 23:48:06.765	3	RM600000.00	COMPLETE
4	6000.0	500.0	3000000.0	2017	2	2023-12-20 23:48:32.346	4	RM2000000.00	COMPLETE
5	4000.0	300.0	1200000.0	2021	3	2023-12-20 23:51:02.614	4	RM4000000.00	COMPLETE
6	4000.0	500.0	200000.0	2006	4	2023-12-21 00:30:52.458	3	RM900000.00	COMPLETE
7	3000.0	300.0	100000.0	2020	2	2023-12-21 00:51:23.242	4	RM15000000.00	COMPLETE
8	3000.0	200.0	600000.0	2020	3	2023-12-22 17:24:38.663	5	RM12000000.00	PENDING

Figure 35: Screenshot of Land List CLI

In land list, it will show all the registered land with their information. It also shows the land registration status, so it can be further be approved in approve land registration function.

```
Land ID #8
Land Area               : 3000
Land Height             : 200
Land Volume             : 600000
Year of Construction    : 2020
Owner ID                : 3
Land Condition (1 - 5)  : 5
Value                   : RM 12000000


Registration of Title Fee: RM100
Select Payment Method
1. Cash
2. Debit Card
3. Credit Card
4. QR Pay
5. Cancel
Enter Payment Method: 1

Enter your password for validation ('X' to cancel): admin123
** Transaction successful. **
** Land Record Created. **
** Successfully registered new land. Please patiently wait for approval. **
```

Figure 36: Screenshot of Login CLI

To register new land, the system will automatically generate the land ID for the new land and admin is required to input all the land information. Then, make transactions for the owner of the land and input password to confirm payment. The land registration and transaction will be sent to pending until admin validate it.

```
Approve Land Registration
--------------------------------------------------
Land ID with pending registration:
LandID # 8
LandID to approve (0 to approve all): 8
** LandID #8 registration approved. **
```

Figure 37: Screenshot of Approve Land Registration CLI

In this function, admin is able to approve land registration that the land registration is pending and has completed transactions. Then, it will reflect the changes to relevant files.

3.2.4 Manage Transaction

```
Manage Transaction
--------------------------------------------------
1. Transaction List
2. Transaction Approval
3. Back
Enter your choice: 2
```

Figure 38: Screenshot of Login CLI

In manage transaction, admin can access to all transaction information and approve pending transaction.

Transaction Information

ID	Land ID	Buyer	Seller	Amount	Pay Method	Type	Recorded Date	Status
1	1	2	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-20 23:46:28.597	COMPLETE
2	2	3	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-20 23:47:07.653	COMPLETE
3	3	4	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-20 23:48:31.067	COMPLETE
4	4	2	0	100.0	CREDITCARD	REGISTRATIONOFTITLE	2023-12-20 23:49:19.114	COMPLETE
5	1	2	0	100.0	QRPAY	REGISTRATIONOFDEEDS	2023-12-20 23:50:10.374	COMPLETE
6	5	3	0	100.0	DEBITCARD	REGISTRATIONOFTITLE	2023-12-20 23:51:37.089	COMPLETE
7	1	3	2	800200.0	CREDITCARD	CONVEYANCE	2023-12-20 23:51:53.804	COMPLETE
8	6	4	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-21 00:32:16.397	COMPLETE
9	4	2	0	100.0	CASH	REGISTRATIONOFDEEDS	2023-12-21 00:46:12.722	COMPLETE
10	7	2	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-21 00:51:51.495	COMPLETE
11	7	2	0	100.0	CREDITCARD	REGISTRATIONOFDEEDS	2023-12-21 00:53:19.613	COMPLETE
12	1	3	0	100.0	CASH	REGISTRATIONOFDEEDS	2023-12-21 00:56:06.572	COMPLETE
13	8	3	0	100.0	CASH	REGISTRATIONOFTITLE	2023-12-22 17:25:47.078	PENDING

Figure 39: Screenshot of Transaction List CLI

Transaction List function displays all the transaction made by users including the transaction status, that required the approval of admin to be complete.

```
Transaction Approval
--------------------------------------------------
Transaction IDs with pending status:
Transaction ID # 13
Transaction ID to approve: 13
** Transaction ID #13 approved. **
```

Figure 40: Screenshot of Transaction Approval CLI

In this function, admin is able to approve transaction that is pending and reflect the changes to relevant files.

3.2.5 Land Record

Land Record

RecID	LandID	OwnerID	TransID	TranStatus	LandStatus	RegStatus
1	1	2	1	COMPLETE	OWNED	COMPLETE
2	2	3	2	COMPLETE	OWNED	COMPLETE
3	3	4	3	COMPLETE	OWNED	COMPLETE
4	4	2	4	COMPLETE	OWNED	COMPLETE
5	1	2	5	COMPLETE	ONSALE	COMPLETE
6	5	3	6	COMPLETE	OWNED	COMPLETE
7	1	3	7	COMPLETE	OWNED	COMPLETE
8	6	4	8	COMPLETE	OWNED	COMPLETE
9	4	2	9	COMPLETE	ONSALE	COMPLETE
10	7	2	10	COMPLETE	OWNED	COMPLETE
11	7	2	11	COMPLETE	ONSALE	COMPLETE
12	1	3	12	COMPLETE	ONSALE	COMPLETE
13	8	3	13	COMPLETE	OWNED	COMPLETE

Onsale Land

ID	Area	Height	Volume	Year	Owner	Registered Date	Condition	Value	Status
1	4000.0	300.0	120000.0	2020	3	2023-12-20 23:46:07.281	4	800000.0	COMPLETE
4	6000.0	500.0	3000000.0	2017	2	2023-12-20 23:48:32.346	4	2000000.0	COMPLETE
7	3000.0	300.0	100000.0	2020	2	2023-12-21 00:51:23.242	4	1.5E7	COMPLETE

Figure 41: Screenshot of Land Record CLI

In land record function, it shows all the land records that has been made and also lands that are currently on sale in the market.

3.3 Customer

3.3.1 Menu

```
Welcome, joshua!

Menu
-----------------------------------------------------
1. Modify Profile
2. Register Land
3. User Land Record
4. Buy Land
5. Logout
Enter your choice:
```

Figure 42: Screenshot of Customer Menu CLI

Customer menu is the landing page upon login successfully as customer. Customers can access to features such as modify profile, register land, user land record and buy land.

3.3.2 Modify Profile

```
New username            : joshie
New password            : joshie123
New age                 : 25
New email               : joshie@gmail.com
New phone number        : 0192836192
New occupation          : Lawyer


** User modified successfully. **
```

Figure 43: Screenshot of Modify User CLI

Customer can modify profile by inputting new information to update the user information.

3.3.3 Register Land

```
Land ID #8
Land Area               : 3000
Land Height             : 200
Land Volume             : 600000
Year of Construction    : 2020
Owner ID                : 3
Land Condition (1 - 5)  : 5
Value                   : RM 12000000


Registration of Title Fee: RM100
Select Payment Method
1. Cash
2. Debit Card
3. Credit Card
4. QR Pay
5. Cancel
Enter Payment Method: 1

Enter your password for validation ('X' to cancel): admin123
** Transaction successful. **
** Land Record Created. **
** Successfully registered new land. Please patiently wait for approval. **
```

Figure 44: Screenshot of Register Land CLI

To register new land, the system will automatically generate the land ID for the new land and customer is required to input all the land information. Then, make transactions and input password to confirm payment. The land registration and transaction will be sent to pending until admin validate it.

3.3.4 User Land Record

```
User Land Record
--------------------------------------------------


                                        My Onsale Land
-------------------------------------------------------------------------------------------------------------------------
ID   | Area      | Height    | Volume     | Year  | Owner  | Registered Date           | Condition  | Value           | Status
-------------------------------------------------------------------------------------------------------------------------
1    | 4000.0    | 300.0     | 120000.0   | 2020  | 3      | 2023-12-20 23:46:07.281   | 4          | 800000.0        | COMPLETE
5    | 4000.0    | 300.0     | 1200000.0  | 2021  | 3      | 2023-12-20 23:51:02.614   | 4          | 4000000.0       | COMPLETE
-------------------------------------------------------------------------------------------------------------------------


                                        My Owned Land
-------------------------------------------------------------------------------------------------------------------------
ID   | Area      | Height    | Volume     | Year  | Owner  | Registered Date           | Condition  | Value           | Status
-------------------------------------------------------------------------------------------------------------------------
2    | 3000.0    | 500.0     | 1500000.0  | 2023  | 3      | 2023-12-20 23:46:41.773   | 5          | 2000000.0       | COMPLETE
8    | 3000.0    | 200.0     | 600000.0   | 2020  | 3      | 2023-12-22 17:24:38.663   | 5          | 1.2E7           | COMPLETE
-------------------------------------------------------------------------------------------------------------------------
Do you want to put any land on sale? (Y/N): Y
Enter Land ID to put on sale (0 to cancel): 2


Registration of Deeds Fee: RM100
Select Payment Method
1. Cash
2. Debit Card
3. Credit Card
4. QR Pay
5. Cancel
Enter Payment Method: 1

Enter your password for validation ('X' to cancel): myles456
** Transaction successful. **
** Land Record Created. **
** Land #2 has been put on sale. **
```

Figure 45: Screenshot of User Land Record CLI

In user land record, it shows all the customer's registered land and filtered out the on-sale land if there is any. The customer can select land to put on sale and make payment for the processing fee. Once the transaction has been made, the customer has to wait for the transaction being approved by admin to register the land to be on sale.

3.3.5 Buy Land

```
Buy Land
--------------------------------------------------


                                          Onsale Land
-------------------------------------------------------------------------------------------------------------------------------
ID   | Area      | Height    | Volume    | Year  | Owner  | Registered Date           | Condition  | Value           | Status
-------------------------------------------------------------------------------------------------------------------------------
1    | 4000.0    | 300.0     | 120000.0  | 2020  | 3      | 2023-12-20 23:46:07.281   | 4          | 800000.0        | COMPLETE
5    | 4000.0    | 300.0     | 1200000.0 | 2021  | 3      | 2023-12-20 23:51:02.614   | 4          | 4000000.0       | COMPLETE
-------------------------------------------------------------------------------------------------------------------------------
Do you want to buy any land? (Y/N): Y
Enter Land ID to buy (0 to cancel): 1


Conveyance Fee  : RM200
Land Value      : RM800000.0
** Total        : RM800200.0 **

Select Payment Method
1. Cash
2. Debit Card
3. Credit Card
4. QR Pay
5. Cancel
Enter Payment Method: 3

Enter your password for validation ('X' to cancel): chenxi123
** Transaction successful. **
** Land Record Created. **
** Land #1 has been purchased. Wait for approval. **
```

Figure 46: Screenshot of Buy Land CLI

The customer can access to the land market where it displays all the lands that are currently on sale for all. Then the user can input the land ID to buy the land and make payment. After payment has been made, the customer wait for the admin to validate the transaction and make landowner transference.

4.0 System Evaluation

4.1 Data Security

One of the main concerns in the field of IT has been data security. The use of hashing algorithms like sha-256 and blockchain technology has significantly increased data security. Data integrity and authenticity are safeguarded by the sha-256 hashing technique, which forbids unauthorised changes by clients or organisations. This includes product information. To protect user privacy and user data from unauthorized program intrusion, the suggested method also employs encryption. In this proposed system, each user is given a public key and a private key. As soon as the user logs in and selects the add product option, the system uses their private key to digitally sign and encrypt the product data. Subsequently, both the encrypted and original, unencrypted product data are stored together. To verify data integrity, the system uses the user's public key to compare the decrypted data to the original product record before adding the pair to the product database. By following this process, unauthorized users cannot add fake product entries, protecting the integrity of the database.

4.2 Data Transparency

Using blockchain technology also gives businesses better data transparency. Through the use of a distributed ledger, users of the proposed system can concurrently access and evaluate product data from any location at any time. Each product record is stored within a block that is linked to other blocks in a blockchain, and all users of the proposed system have access to a copy of the shared ledger containing the product data. This shared ledger significantly improves data transparency within the system by enabling all stakeholders, including customers and registered users, to easily monitor details about the product, such as its name, arrival and departure times, and locations. Furthermore, since every block has a hash that came before it, changing one block would mean changing every block that followed, making data manipulation much more difficult. By doing this, users can prevent unauthorized changes and keep accurate and reliable data inside the program.

4.3 Data Accessibility

To enhance data accessibility in the land registration system, incorporating digital signature authentication is crucial for approving transactions. The use of digital signatures enhances security and ensures that each transaction is validated and authorized by the rightful parties, adding a layer of trust to the approval process. This implementation provides a strong mechanism to verify both the authenticity of involved parties and the integrity of transaction data. This method not only improves the trustworthiness of the approval procedure but also ensures the security of important data. Authorized parties can securely access and process approved transactions, confident that the authenticity of the data has been confirmed through digital signatures. This not only simplifies the approval process but also assures that the system's data remains unaltered and protected, thereby enabling effective and dependable data accessibility within the land registration framework. In the future, there is potential to expand the application of digital signature to other areas of the system, such as land registration approval. By broadening the use of digital signature verification, it could enhance accessibility and trust in different transactional procedures. This would help create a comprehensive and secure ecosystem for land registration. Such expansion would strengthen the system's dedication to preserving data integrity, laying a strong groundwork for future advancements.

5.0 Conclusion

The group assignment is on the business of land registration, specifically on the full procedure of the land registration. To keep up with the ever-growing industries in technology, land registration can be easily done on the computer compared to using pen and paper. Not only does it help make the process more efficient, it also helps increase the security of the data so that its not tampered with and can be traced back to their original owner. Blockchain technology has all the essential needs to make sure that the data of each land and their owners are safe and non-editable in a secure ledger to ensure its authenticity.

The suggested blockchain system that was developed, aims to store the data of each land data, user's personal data and land's owner while assuring data security and traceability. The distributed ledger system and immutability of blockchain technology provide a transparent and reliable platform for all parties involved in the land registration and transferring of ownership. The process flowchart and system architecture diagram demonstrate how the blockchain solution can track and record user information, previous owners of specific lands, and land data. By putting this solution into practice, product traceability will be strengthened, and overall productivity will rise as suppliers and customers will find it easier to track past and current landowners as they travel through the many users that buy or register lands.

In conclusion, the project examined the system for allowing a simpler approach to registering lands and transfer of ownership to other users, while highlighting the significance of traceability, and analyse data that is crucial for the creation of the blockchain solution. The blockchain technology comes packaged with many features like improved data security, authenticity, and accountability. The adoption for blockchain technology in the land registry area can establish a more streamlined process and efficient experience for all users in need of a secure, trustworthy, and flawless solutions.

6.0 References

Coinbase. (2022). Why digital signatures are essential for blockchains. Coinbase. https://www.coinbase.com/en-gb/cloud/discover/dev-foundations/digital-signatures

Dremio. (n.d.). *Data Immutability*. https://www.dremio.com/wiki/data-immutability/

IBM. (n.d.). *What is Blockchain Technology?* https://www.ibm.com/topics/blockchain

Krishnapriya, S., & Sarath, G. (2020). Securing Land Registration using Blockchain. *Procedia Computer Science*, *171*, 1708–1715. https://doi.org/10.1016/j.procs.2020.04.183

Sajana, P., Sindhu, M., & Sethumadhavan, M. (2018). *On blockchain applications : Hyperledger Fabric and Ethereum*. https://www.semanticscholar.org/paper/On-Blockchain-Applications-%3A-Hyperledger-Fabric-And-TIFAC-CORE/767410f40ed2ef1b8b759fec3782d8a0f2f8ad40

Table of Figures