perplexity

QuantumLedger: Java-Only Blockchain Accounting System

Project Overview

QuantumLedger is a comprehensive Java-based blockchain accounting system designed to leverage distributed ledger technology for secure, transparent, and immutable financial record-keeping[1] [2]. The system combines traditional accounting principles with blockchain technology to create a robust platform for transaction management, user authentication, and data persistence[3] [4].

Project Architecture

Core Components

The QuantumLedger system is built around four primary modules that work together to provide a complete blockchain accounting solution[1] [5]:

1. **Block Module**: Contains the fundamental blockchain data structure with timestamp, transaction data, previous hash, and current hash
2. **Blockchain Module**: Implements a LinkedList-based chain of blocks for maintaining the distributed ledger
3. **LedgerService Module**: Handles transaction addition and retrieval operations
4. **UserAuth Module**: Provides simple user authentication and role-based access control

Technology Stack

The project utilizes a carefully selected technology stack optimized for Java development[6] [7]:

- **Java Security**: `java.security.MessageDigest` for SHA-256 cryptographic hashing[6] [8]
- **Data Persistence**: JDBC with SQLite for lightweight, embedded database operations[9] [7]
- **Framework**: Maven-based project structure for dependency management and build automation[10] [11]
- **ORM**: JPA (Java Persistence API) for object-relational mapping[12] [13]

Maven Project Structure

Directory Layout

Following Maven's standard directory structure[10] [14]:

```
quantum-ledger/
├── pom.xml
├── README.md
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── quantumledger/
│   │   │           ├── blockchain/
│   │   │           │   ├── Block.java
│   │   │           │   └── Blockchain.java
│   │   │           ├── service/
│   │   │           │   └── LedgerService.java
│   │   │           ├── auth/
│   │   │           │   └── UserAuth.java
│   │   │           ├── entity/
│   │   │           │   ├── User.java
│   │   │           │   └── Transaction.java
│   │   │           └── util/
│   │   │               └── CryptoUtil.java
│   │   └── resources/
│   │       ├── META-INF/
│   │       │   └── persistence.xml
│   │       └── application.properties
│   └── test/
│       └── java/
│           └── com/
│               └── quantumledger/
│                   └── BlockchainTest.java
└── target/
```

Maven POM Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.quantumledger</groupId>
    <artifactId>quantum-ledger-blockchain</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>QuantumLedger Blockchain Accounting</name>
    <description>Java-based blockchain accounting system with JPA and SQLite</descriptio
```

```xml
    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <hibernate.version>5.6.15.Final</hibernate.version>
        <sqlite.version>3.42.0.0</sqlite.version>
    </properties>

    <dependencies>
        <!-- JPA/Hibernate Dependencies -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <!-- SQLite Database Driver -->
        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>${sqlite.version}</version>
        </dependency>

        <!-- SQLite Hibernate Dialect -->
        <dependency>
            <groupId>com.github.gwenn</groupId>
            <artifactId>sqlite-dialect</artifactId>
            <version>0.1.2</version>
        </dependency>

        <!-- JSON Processing -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.10.1</version>
        </dependency>

        <!-- Testing Dependencies -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Database Design and Schema

JPA Entity Configuration

The database schema is designed to support blockchain operations while maintaining relational integrity[15] [16]. The system uses JPA annotations to map Java objects to database tables[12] [13]:

Block Entity

```
@Entity
@Table(name = "blocks")
public class Block {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "block_index", nullable = false)
    private int index;

    @Column(name = "timestamp", nullable = false)
    private long timestamp;

    @Column(name = "previous_hash", length = 64)
    private String previousHash;

    @Column(name = "current_hash", length = 64, nullable = false)
    private String hash;

    @Column(name = "data", columnDefinition = "TEXT")
    private String data;

    @Column(name = "nonce")
    private int nonce;

    // Constructors, getters, and setters
}
```

Transaction Entity

```
@Entity
@Table(name = "transactions")
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "from_address", nullable = false)
    private String fromAddress;

    @Column(name = "to_address", nullable = false)
    private String toAddress;
```

```java
    @Column(name = "amount", nullable = false)
    private BigDecimal amount;

    @Column(name = "timestamp", nullable = false)
    private long timestamp;

    @Column(name = "transaction_hash", length = 64, unique = true)
    private String transactionHash;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "block_id")
    private Block block;

    // Constructors, getters, and setters
}
```

User Entity

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username", unique = true, nullable = false)
    private String username;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Enumerated(EnumType.STRING)
    @Column(name = "role", nullable = false)
    private UserRole role;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    // Constructors, getters, and setters
}
```

Database Schema

The SQLite database schema supports the core blockchain functionality [17] [16]:

```sql
-- Blocks table for storing blockchain blocks
CREATE TABLE blocks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    block_index INTEGER NOT NULL,
    timestamp INTEGER NOT NULL,
    previous_hash VARCHAR(64),
    current_hash VARCHAR(64) NOT NULL UNIQUE,
```

```
    data TEXT,
    nonce INTEGER DEFAULT 0,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Transactions table for storing individual transactions
CREATE TABLE transactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    from_address VARCHAR(255) NOT NULL,
    to_address VARCHAR(255) NOT NULL,
    amount DECIMAL(15,2) NOT NULL,
    timestamp INTEGER NOT NULL,
    transaction_hash VARCHAR(64) UNIQUE,
    block_id INTEGER,
    FOREIGN KEY (block_id) REFERENCES blocks(id)
);

-- Users table for authentication and authorization
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('ADMIN', 'USER', 'AUDITOR')),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance optimization
CREATE INDEX idx_blocks_hash ON blocks(current_hash);
CREATE INDEX idx_blocks_prev_hash ON blocks(previous_hash);
CREATE INDEX idx_transactions_hash ON transactions(transaction_hash);
CREATE INDEX idx_transactions_addresses ON transactions(from_address, to_address);
CREATE INDEX idx_users_username ON users(username);
```

JPA Configuration

The `persistence.xml` file configures JPA with SQLite[9] [7]:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">

    <persistence-unit name="quantumLedgerPU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <class>com.quantumledger.entity.Block</class>
        <class>com.quantumledger.entity.Transaction</class>
        <class>com.quantumledger.entity.User</class>

        <properties>
            <!-- Database connection properties -->
            <property name="javax.persistence.jdbc.driver" value="org.sqlite.JDBC"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:sqlite:quantum_ledger
```

```
            <!-- Hibernate properties -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.SQLiteDialect
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>

            <!-- SQLite specific properties -->
            <property name="hibernate.connection.isolation" value="2"/>
            <property name="hibernate.connection.release_mode" value="after_transaction"/
        </properties>
    </persistence-unit>
</persistence>
```

Core Implementation

Block Implementation

The Block class represents individual blocks in the blockchain[1] [5]:

```
package com.quantumledger.blockchain;

import javax.persistence.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.nio.charset.StandardCharsets;

@Entity
@Table(name = "blocks")
public class Block {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int index;
    private long timestamp;
    private String previousHash;
    private String hash;
    private String data;
    private int nonce;

    public Block(int index, String data, String previousHash) {
        this.index = index;
        this.data = data;
        this.previousHash = previousHash;
        this.timestamp = System.currentTimeMillis();
        this.nonce = 0;
        this.hash = calculateHash();
    }

    public String calculateHash() {
        String input = index + timestamp + previousHash + data + nonce;
        return sha256(input);
    }
```

```java
    private String sha256(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(input.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();

            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) {
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("SHA-256 algorithm not available", e);
        }
    }

    public void mineBlock(int difficulty) {
        String target = new String(new char[difficulty]).replace('\0', '0');
        while (!hash.substring(0, difficulty).equals(target)) {
            nonce++;
            hash = calculateHash();
        }
    }

    // Getters and setters
}
```

Blockchain Implementation

The Blockchain class manages the chain of blocks using a LinkedList structure[1] [5]:

```java
package com.quantumledger.blockchain;

import java.util.LinkedList;
import java.util.List;

public class Blockchain {
    private LinkedList<Block> chain;
    private int difficulty;

    public Blockchain() {
        this.chain = new LinkedList<>();
        this.difficulty = 4; // Adjustable mining difficulty
        createGenesisBlock();
    }

    private void createGenesisBlock() {
        Block genesis = new Block(0, "Genesis Block", "0");
        genesis.mineBlock(difficulty);
        chain.add(genesis);
    }
```

```
    public Block getLatestBlock() {
        return chain.getLast();
    }

    public void addBlock(Block newBlock) {
        newBlock.setPreviousHash(getLatestBlock().getHash());
        newBlock.mineBlock(difficulty);
        chain.add(newBlock);
    }

    public boolean isChainValid() {
        for (int i = 1; i < chain.size(); i++) {
            Block currentBlock = chain.get(i);
            Block previousBlock = chain.get(i - 1);

            if (!currentBlock.getHash().equals(currentBlock.calculateHash())) {
                return false;
            }

            if (!currentBlock.getPreviousHash().equals(previousBlock.getHash())) {
                return false;
            }
        }
        return true;
    }

    public List<Block> getChain() {
        return new LinkedList<>(chain);
    }
}
```

LedgerService Implementation

The LedgerService handles transaction operations and blockchain management [2] [4]:

```
package com.quantumledger.service;

import com.quantumledger.blockchain.Block;
import com.quantumledger.blockchain.Blockchain;
import com.quantumledger.entity.Transaction;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.math.BigDecimal;
import java.util.List;

public class LedgerService {
    private Blockchain blockchain;
    private EntityManagerFactory emf;

    public LedgerService() {
        this.blockchain = new Blockchain();
        this.emf = Persistence.createEntityManagerFactory("quantumLedgerPU");
    }
```

```
    public void addTransaction(String fromAddress, String toAddress, BigDecimal amount) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();

            // Create transaction entity
            Transaction transaction = new Transaction();
            transaction.setFromAddress(fromAddress);
            transaction.setToAddress(toAddress);
            transaction.setAmount(amount);
            transaction.setTimestamp(System.currentTimeMillis());
            transaction.setTransactionHash(generateTransactionHash(transaction));

            // Create new block with transaction data
            String transactionData = createTransactionData(transaction);
            Block newBlock = new Block(blockchain.getChain().size(), transactionData,
                                       blockchain.getLatestBlock().getHash());

            // Add block to blockchain
            blockchain.addBlock(newBlock);

            // Persist to database
            em.persist(transaction);
            persistBlock(em, newBlock);

            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw new RuntimeException("Failed to add transaction", e);
        } finally {
            em.close();
        }
    }

    public List<Transaction> getTransactionHistory() {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createQuery("SELECT t FROM Transaction t ORDER BY t.timestamp DESC"
                                  Transaction.class).getResultList();
        } finally {
            em.close();
        }
    }

    public BigDecimal getBalance(String address) {
        EntityManager em = emf.createEntityManager();
        try {
            BigDecimal incoming = em.createQuery(
                "SELECT COALESCE(SUM(t.amount), 0) FROM Transaction t WHERE t.toAddress =
                BigDecimal.class)
                .setParameter("address", address)
                .getSingleResult();

            BigDecimal outgoing = em.createQuery(
                "SELECT COALESCE(SUM(t.amount), 0) FROM Transaction t WHERE t.fromAddress
```

```
                BigDecimal.class)
                .setParameter("address", address)
                .getSingleResult();

            return incoming.subtract(outgoing);
        } finally {
            em.close();
        }
    }

    private String generateTransactionHash(Transaction transaction) {
        // Implementation using MessageDigest for transaction hashing
        String data = transaction.getFromAddress() + transaction.getToAddress() +
                     transaction.getAmount() + transaction.getTimestamp();
        return sha256(data);
    }

    // Additional helper methods...
}
```

UserAuth Implementation

The UserAuth class provides authentication and role-based access control[18] [19]:

```
package com.quantumledger.auth;

import com.quantumledger.entity.User;
import com.quantumledger.entity.UserRole;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.nio.charset.StandardCharsets;

public class UserAuth {
    private EntityManagerFactory emf;
    private User currentUser;

    public UserAuth() {
        this.emf = Persistence.createEntityManagerFactory("quantumLedgerPU");
    }

    public boolean authenticate(String username, String password) {
        EntityManager em = emf.createEntityManager();
        try {
            User user = em.createQuery("SELECT u FROM User u WHERE u.username = :username
                         .setParameter("username", username)
                         .getSingleResult();

            String hashedPassword = hashPassword(password);
            if (user.getPasswordHash().equals(hashedPassword)) {
                this.currentUser = user;
                return true;
            }
```

```
        } catch (Exception e) {
            // User not found or other error
        } finally {
            em.close();
        }
        return false;
    }

    public boolean hasPermission(String operation) {
        if (currentUser == null) {
            return false;
        }

        switch (currentUser.getRole()) {
            case ADMIN:
                return true; // Admin has all permissions
            case AUDITOR:
                return operation.equals("VIEW") || operation.equals("AUDIT");
            case USER:
                return operation.equals("VIEW") || operation.equals("ADD_TRANSACTION");
            default:
                return false;
        }
    }

    public void createUser(String username, String password, UserRole role) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();

            User user = new User();
            user.setUsername(username);
            user.setPasswordHash(hashPassword(password));
            user.setRole(role);
            user.setCreatedAt(java.time.LocalDateTime.now());

            em.persist(user);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw new RuntimeException("Failed to create user", e);
        } finally {
            em.close();
        }
    }

    private String hashPassword(String password) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(password.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();

            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) {
                    hexString.append('0');
```

```
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("SHA-256 algorithm not available", e);
        }
    }

    public User getCurrentUser() {
        return currentUser;
    }

    public void logout() {
        this.currentUser = null;
    }
}
```

Benefits and Applications

Blockchain Accounting Advantages

The QuantumLedger system provides several key benefits over traditional accounting systems[20] [4]:

1. **Enhanced Security**: Cryptographic hashing and immutable record-keeping prevent data tampering and fraud[21] [22]
2. **Improved Transparency**: All authorized participants have access to the same ledger, providing clear audit trails[20] [23]
3. **Real-time Auditing**: Transactions can be verified and audited in real-time, reducing compliance costs[22] [23]
4. **Reduced Costs**: Automated processes and reduced reconciliation efforts lead to significant cost savings[21] [23]
5. **Data Integrity**: Blockchain's immutable nature ensures that once recorded, transactions cannot be altered[4] [22]

Use Cases

The system is particularly well-suited for[24] [25]:

- **Small to Medium Enterprises**: Simplified accounting processes with enhanced security[3] [25]
- **Internal Audits**: Real-time verification and tamper-proof records[22] [23]
- **Supply Chain Finance**: Transparent tracking of financial flows across multiple parties[20] [4]
- **Regulatory Compliance**: Automated reporting and immutable audit trails[22] [23]

Security Considerations

Cryptographic Security

The system implements robust security measures[6] [26]:

- **SHA-256 Hashing**: Industry-standard cryptographic hashing for data integrity[6] [8]
- **Digital Signatures**: Transaction authentication using cryptographic signatures[27] [28]
- **Access Control**: Role-based permissions for different user types[18] [19]
- **Data Encryption**: Sensitive data protection through encryption techniques[21] [26]

Best Practices

Following Java blockchain security best practices[26] [25]:

1. **Input Validation**: All user inputs are validated and sanitized to prevent injection attacks[26]
2. **Secure Key Management**: Proper handling and storage of cryptographic keys[26]
3. **Regular Security Audits**: Continuous monitoring and assessment of system security[26]
4. **Multi-factor Authentication**: Enhanced user authentication mechanisms[26]

Development and Testing

Testing Strategy

The project includes comprehensive testing approaches[1] [5]:

```
package com.quantumledger;

import com.quantumledger.blockchain.Block;
import com.quantumledger.blockchain.Blockchain;
import org.junit.Test;
import static org.junit.Assert.*;

public class BlockchainTest {

    @Test
    public void testBlockchainCreation() {
        Blockchain blockchain = new Blockchain();
        assertNotNull(blockchain);
        assertEquals(1, blockchain.getChain().size());
        assertTrue(blockchain.isChainValid());
    }

    @Test
    public void testBlockAddition() {
        Blockchain blockchain = new Blockchain();
        Block newBlock = new Block(1, "Test Transaction", blockchain.getLatestBlock().get
        blockchain.addBlock(newBlock);
```

```
        assertEquals(2, blockchain.getChain().size());
        assertTrue(blockchain.isChainValid());
    }

    @Test
    public void testChainValidation() {
        Blockchain blockchain = new Blockchain();

        // Add valid blocks
        blockchain.addBlock(new Block(1, "Transaction 1", blockchain.getLatestBlock().get
        blockchain.addBlock(new Block(2, "Transaction 2", blockchain.getLatestBlock().get

        assertTrue(blockchain.isChainValid());

        // Tamper with a block
        blockchain.getChain().get(1).setData("Tampered Data");
        assertFalse(blockchain.isChainValid());
    }
}
```

Performance Optimization

The system is designed for optimal performance[29] [26]:

- **Efficient Data Structures**: LinkedList implementation for blockchain operations[1] [5]
- **Database Indexing**: Strategic indexes on frequently queried fields[16] [30]
- **Connection Pooling**: Optimized database connection management[9] [7]
- **Caching Strategies**: In-memory caching for frequently accessed data[29] [26]

Future Enhancements

Scalability Improvements

Potential enhancements for enterprise deployment[17] [29]:

1. **Distributed Network Support**: Multi-node blockchain network implementation[17] [29]
2. **Smart Contracts**: Automated contract execution capabilities[22] [24]
3. **Advanced Consensus Mechanisms**: Implementation of Proof-of-Stake or other consensus algorithms[5] [31]
4. **Integration APIs**: REST APIs for third-party system integration[11] [25]

Advanced Features

Additional functionality for comprehensive accounting systems[23] [32]:

- **Regulatory Reporting**: Automated compliance reporting features[22] [23]
- **Multi-currency Support**: Support for multiple currencies and exchange rates[25] [23]
- **Advanced Analytics**: Business intelligence and reporting capabilities[2] [23]

- **Mobile Applications**: Mobile interfaces for remote access[24] [32]

Conclusion

QuantumLedger represents a significant advancement in blockchain-based accounting systems, combining the security and transparency of distributed ledger technology with the reliability and performance of Java enterprise development[1] [4]. The system's modular architecture, comprehensive security features, and robust database design make it suitable for a wide range of accounting applications, from small business bookkeeping to enterprise-level financial management[3] [25].

The implementation demonstrates how blockchain technology can be effectively integrated into traditional accounting workflows, providing enhanced security, improved transparency, and reduced operational costs while maintaining compatibility with existing business processes[20] [23]. As blockchain technology continues to evolve, QuantumLedger provides a solid foundation for future enhancements and enterprise-scale deployments[17] [32].

⁂

1. https://www.baeldung.com/java-blockchain
2. https://www.cryptoworth.com/blog/blockchain-accounting-software
3. https://kilthub.cmu.edu/articles/thesis/Reimagining_Triple-Entry_Accounting_A_Blockchain_System_for_Confidential_Transaction_Verification/29088194
4. https://www.invensis.net/blog/impact-of-blockchain-on-accounting
5. https://github.com/dkrizhanovskyi/blockchain-on-java
6. https://www.baeldung.com/sha-256-hashing-java
7. https://dominoc925.blogspot.com/2015/11/an-example-of-using-sqlite-database-in.html
8. https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html
9. https://stackoverflow.com/questions/906241/jpasqlite-problem
10. https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
11. https://github.com/blockchain/api-v1-client-java
12. https://www.scribd.com/document/272694727/annotations-mapping-docx
13. https://www.javacodegeeks.com/2013/04/jpa-determining-the-owning-side-of-a-relationship.html
14. https://stackoverflow.com/questions/44942030/eclipse-maven-project-structure
15. https://www.dbdesigner.net/blockchain-database-schema-revolutionize-decentralized-systems/
16. https://github.com/dr-blockchains/blockchain-database
17. https://www.mongodb.com/resources/basics/databases/blockchain-database
18. https://github.com/Abhinav0915/RoleBasedAuthentication
19. https://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/Using_JBoss_Login_Modules-UsersRolesLoginModule.html
20. https://www.lsbf.org.uk/blog/online-learning/what-is-the-role-of-blockchain-in-accounting
21. https://www.ibm.com/think/topics/benefits-of-blockchain

22. https://binus.ac.id/bekasi/2024/07/blockchain-revolutionizes-accounting-and-enhances-security-efficiency-and-transparency/
23. https://www.businesstechweekly.com/finance-and-accounting/blockchain/blockchain-for-accounting/
24. https://www.startupindia.gov.in/content/sih/en/bloglist/blogs/Blockchain_Applications.html
25. https://pubmed.ncbi.nlm.nih.gov/38324616/
26. https://www.bydfi.com/en/questions/how-can-i-use-java-to-build-secure-and-efficient-blockchain-solutions
27. https://stackoverflow.com/questions/5531455/how-to-hash-some-string-with-sha-256-in-java
28. https://www.w3docs.com/snippets/java/how-to-hash-some-string-with-sha-256-in-java.html
29. https://www.vldb.org/pvldb/vol15/p1092-loghin.pdf
30. https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-database-ledger?view=sql-server-ver17
31. https://github.com/AbdullahAbdelglil/Java-Based-Blockchain
32. https://en.wikipedia.org/wiki/Blockchain