ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Département : Mathématique et Informatique.

Filière : Ingénierie informatique, Big Data et Cloud Computing (II-BDCC)

Examen blanc

Blockchain

DIOP AMARY
II-BDCC3

Enoncé

Première partie : Micro-Service

Créer le micro-service blockchain-service permettant d'implémenter les bases de la blockchaine

1. Créer la couche DAO
    a. Créer les entités JPA
    b. Créer les interfaces basées sur Spring Data
    c. Tester la couche DAO
2. Créer la couche service
    a. Créer l'interface BlockService et son implémentation
    b. Créer l'interface Blockchain service et son implémentation
    c. Tester la couche service
3. Créer la couche WEB
    a. Créer les Rest Controllers
    b. Tester les API REST
4. Sécuriser le Micro-service en utilisant Keycloak

1. Créer la couche DAO

a. Créer les entités JPA

```java
@Entity @Data @AllArgsConstructor @NoArgsConstructor
public class BlockChainEntity {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private int difficulte;
    private double miningReward;
    @OneToMany(mappedBy = "blockchain", fetch = FetchType.EAGER)
    private Collection<BlockEntity> blocks = new ArrayList<BlockEntity>();

    public BlockChainEntity(String nom, int diff, double mR){
        this.nom = nom;
        this.difficulte = diff;
        this.miningReward = mR;
    }
}
```

```java
@Entity @Data @AllArgsConstructor @NoArgsConstructor
public class BlockEntity {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @CreatedDate @Temporal(TemporalType.TIMESTAMP)
    private Date dateBlock;
    private String hash;
    private String hashPrevious;
    private int nonce;
    @OneToMany(mappedBy = "block", fetch = FetchType.EAGER)
    private Collection<TransactionEntity> listeTransactions = new ArrayList<TransactionEntity>();
    @ManyToOne
    private BlockChainEntity blockchain;

    public BlockEntity(String hash, String hashP, int nonce){
        this.hash = hash;
        this.hashPrevious = hashP;
        this.nonce = nonce;
    }

    @PrePersist
    public void setCreatedAt() { this.dateBlock = new Date(); }
}
```

```java
@Entity @AllArgsConstructor @NoArgsConstructor @Data
public class TransactionEntity {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @CreatedDate @Temporal(TemporalType.TIMESTAMP)
    private Date dateTransaction;
    private String adresseSource;
    private String adresseDestination;
    private double montantTransaction;
    @ManyToOne
    private BlockEntity block;

    public TransactionEntity(String source, String dest, double montant){
        this.adresseSource = source;
        this.adresseDestination = dest;
        this.montantTransaction = montant;
    }

    @PrePersist
    public void setCreatedAt() { this.dateTransaction= new Date(); }
}
```

b. Créer les interfaces basées sur Spring Data

c. Tester la couche DAO

BlockRepository.java

```java
import ...

public interface BlockRepository extends JpaRepository<BlockEntity, Long>{
}
```

TransactionRepository.java

```java
import ...

public interface TransactionRepository extends JpaRepository<TransactionEntity, Long> {
}
```

BlockChainRepository.java

```java
import ...

public interface BlockChainRepository extends JpaRepository<BlockChainEntity, Long> {
}
```

Créer la couche service

a. Créer l'interface BlockService et son implémentation

```java
public class SHA256Util {
    private static final Charset UTF_8 = StandardCharsets.UTF_8;

    private static byte[] digest(byte[] input) {
        String algorithm = "SHA-256";
        MessageDigest md;
        try {
            md = MessageDigest.getInstance(algorithm);
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalArgumentException(e);
        }
        return md.digest(input);
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte b : bytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }
```

Opération qui permet de calculer le hash du block

```java
    public static String hash(String data){
        String hash;
        byte[] shaInBytes = SHA256Util.digest(data.getBytes(UTF_8));
        hash = bytesToHex(shaInBytes);
        return hash;
    }
```

```java
@Bean
CommandLineRunner start(TransactionRepository transact
    return arg -> {

        String texte1 = "texte1";
        System.out.println(hash(texte1));
        String texte2 = "texte2";
        System.out.println(hash(texte2));
    };
}
```

Teste

```
f1d5af3f2c6829a4bbbea7df2a510e01fa457b620c4600ca6aa022e91a3a63aa
67f8f3e13cf7303fa2bdc131e41e9fa2c81a29c7092cfddce3f346a543a952be
```

Console

```java
@Override
public void createBlock(String hashPrevious, List<TransactionEntity> transactions) {
    BlockEntity blockEntity = new BlockEntity();
    blockEntity.setHashPrevious(hashPrevious);
    blockEntity.setListeTransactions(transactions);

    String chaine = createStringBuffer(
    blockEntity.setHash(hash(chaine));

    blockRepository.save(blockEntity);
}
```

Create Block avec un hash particulier

```java
return arg -> {
    transactionRepository.save(new TransactionEntity( source: "source1",  dest: "destination1",  montant: 1000));
    transactionRepository.save(new TransactionEntity( source: "source2",  dest: "destination2",  montant: 2000));
    List<TransactionEntity> transactions = transactionRepository.findAll();

    blockService.createBlock( hashPrevious: "hashPrevious", transactions);
    BlockEntity blockEntity = blockRepository.findById(1L).get();
    System.out.println(blockEntity);
```

Teste

```
(id=1, dateBlock=2021-12-04 13:41:40.415, hash=04bf11c0103869a92286ba38f822ba6d246787547bb269761f12e184ea7a9835, hashPrev
```

Console

b. Créer l'interface Blockchain service et son implémentation

c. Tester la couche service

Create Blockchaine

```java
@Override
public void createBlockchain(BlockchaineDto blockChainDto, BlockEntity genisisBlock) {
    BlockChainEntity blockChainEntity = new BlockChainEntity();
    ModelMapper modelMapper = new ModelMapper();
    blockChainEntity = modelMapper.map(blockChainDto, BlockChainEntity.class);
    blockChainEntity.getBlocks().add(genisisBlock);
    blockChainRepository.save(blockChainEntity);
}
```

```
return arg -> {
    transactionRepository.save(new TransactionEntity( source: "source1",  dest: "destination1",  montant: 1000));
    transactionRepository.save(new TransactionEntity( source: "source2",  dest: "destination2",  montant: 2000));
    List<TransactionEntity> transactions = transactionRepository.findAll();

    blockService.createBlock( hashPrevious: "hashPrevious", transactions);
    BlockEntity blockEntity = blockRepository.findById(1L).get();
    System.out.println(blockEntity);

    BlockchaineDto blockchaineDto = new BlockchaineDto( nom: "My-blockchain",  difficulte: 3,  miningReward: 500.0);
    blockchainService.createBlockchain(blockchaineDto, blockEntity);
    BlockChainEntity blockChainEntity = blockChainRepository.findById(1L).get();
    System.out.println(blockChainEntity);
```

Teste

```
BlockEntity(id=1, dateBlock=2021-12-04 15:45:57.335, hash=b712fb3a26e5df9571a97588221d2159896c02dcd6e
BlockChainEntity(id=1, nom=My-blockchain, difficulte=3, miningReward=500.0, blocks=[])
```

Console

```
@Override
public void miner(Long id) {
    BlockEntity blockEntity = blockRepository.findById(id).get();
    List<TransactionEntity> transactions = (List<TransactionEntity>) blockEntity.getListeTransactions();
    String chaine = SHA256Util.createStringBuffer();
    String hash = SHA256Util.hash(chaine);
    String hashHeader = hash.substring(0, 3);
    while(!hashHeader.equals("000")){
        hash = SHA256Util.hash(chaine);
        System.out.println(hash);
        hashHeader = hash.substring(0, 3);
        chaine = SHA256Util.createStringBuffer();
    }
    blockEntity.setHash(hash);
    blockRepository.save(blockEntity);
}
```

Miner un block

```
return arg -> {
    transactionRepository.save(new TransactionEntity( source: "source1",  dest: "destination1",  montant: 1000));
    transactionRepository.save(new TransactionEntity( source: "source2",  dest: "destination2",  montant: 2000));
    List<TransactionEntity> transactions = transactionRepository.findAll();

    blockService.createBlock( hashPrevious: "hashPrevious", transactions);
    BlockEntity blockEntity = blockRepository.findById(1L).get();


    BlockchaineDto blockchaineDto = new BlockchaineDto( nom: "My-blockchain",  difficulte: 3,  miningReward: 500.0);
    blockchainService.createBlockchain(blockchaineDto, blockEntity);
    BlockChainEntity blockChainEntity = blockChainRepository.findById(1L).get();
    blockchainService.miner( id: 1L);

    BlockEntity blockEntityMined = blockRepository.findById(1L).get();
    System.out.println(blockEntityMined);
    System.out.println("----------hash---->  " + blockEntityMined.getHash());
```

Teste

```
ad7d57c88998b9ed6d6707ba02f46efaed001360e141415302c533e9c08e4f96
4894c94d167bba75b6bbaf7efe12a0111fdd3d6e2cbe45354732cb55f7a893a7
15aa367477e9dad9fbb23ac555a44ede578cd1c87b23b578ae9c6d32dbed2958
000cb46905c3635bc8bf1a0167ec94d65ea0cf44d230411fef3d7b13f4c7fcdf
BlockEntity(id=1, dateBlock=2021-12-04 16:31:43.797, hash=000cb46905c3635bc8bf1a0167ec94d65ea0cf44d230411fef3d7b13f4c7fcdf, hashPrev
----------hash---->  000cb46905c3635bc8bf1a0167ec94d65ea0cf44d230411fef3d7b13f4c7fcdf
```

Console