



Instituto Politécnico de Tomar

COMPUTAÇÃO DISTRIBUÍDA

ENGENHARIA INFOMÁTICA

3º ANO 23/24

Relatório de Projeto Final

Election Ledger



Índice



1	Grupo de Trabalho	1
2	Introdução	3
2.1	Estado da arte	3
2.2	Exemplos de aplicações e projetos baseados em blockchain	9
3	Descrição do projeto	11
4	Trabalho 1 - Blockchain.....	12
4.1	Arquitetura da aplicação	12
4.2	Biblioteca electionledger.blockchain.....	13
4.2.1	Classe Block.java	13
4.2.2	Classe BlockChain.java.....	15
4.2.3	Classe BlockchainException.java.....	16
4.2.4	Classe Transactions.java	16
4.2.5	Classe Transfer.java	17
5	Trabalho 2 – Computação Multitarefa	19
5.1	Arquitetura da aplicação	19
	Biblioteca electionledger.miner	19
5.1.1	Classe MinerP2P.java	20
5.1.2	Classe Interior MiningTHR	21
5.1.3	Classe MiningListener.java.....	22
6	Trabalho 3 – Computação Distribuída.....	23
6.1	Arquitetura da aplicação	23
	Biblioteca electionledger.node	24
6.1.1	Classe AutomaticP2P.java	24
6.1.2	Classe RemoteInterface.java	25
6.1.3	Classe RemoteObject .java.....	25
7	Trabalho 4 – Segurança.....	33
7.1	Arquitetura da aplicação	33
	Biblioteca electionledger. utils.....	35
7.1.1	Classe Credentials.java.....	35
8	Interface Gráfica.....	37
8.1	Autenticação de Utilizadores	37
8.1.1	Interfaces	37
8.1.2	Funcionalidades	39
8.2	Minerador da Rede.....	39
8.2.1	Interfaces	39
8.3	Utilizador “Master”	42
8.3.1	Interfaces	42
8.4	Utilizador “Eleitor”	45
8.4.1	Interfaces	45
9	Limitações e Desenvolvimentos Futuros.....	47
10	Conclusão.....	49
11	Referências	50
11.1	Documentos	50

11.2	Referências web.....	50
------	----------------------	----

Figuras

Figura 1 Nome do aluno 1	1
Figura 2 Nome do aluno 2	1
Figura 3 – Blockchain [w1]	3
Figura 4 Algumas das principais descobertas na timeline da blockchain.	4
Figura 5 Header de um bloco da blockchain com todos os campos básicos.	4
Figura 6 Representação de uma Merkle Tree	5
Figura 7 - Diagrama de classes do package BlockChain	12
Figura 8 - Diagrama de classes do package Miner	19
Figura 9 - Diagrama de classes do package node	23
Figura 10 - Diagrama de classes do package utils	33
Figura 11 Interface de Login.....	37
Figura 12 Interface de registo.....	38
Figura 13 Interface Miner com bloco minado.	39
Figura 14 Interface de miner com informação de transações.....	40
Figura 15 Interface de Miner para conexões.	40
Figura 16 Interface de Miner com conexões na rede.	41
Figura 17Interface da comissão eleitoral com o controlo das fases da eleição.	42
Figura 18 Interface de comissão eleitoral que apresenta a informação dos eleitores.	43
Figura 19Interface da comissão eleitoral que permite explorar os blocos da blockchain.	43
Figura 20 Interface da comissão eleitoral que apresenta logs da rede.	44
Figura 21Interface da comissão eleitoral que permite ver informação do utilizador master.	44
Figura 22Interface de eleitor com janela de voto e confirmação.	45
Figura 23 interface de eleitor com resultados da eleição	46

1 Grupo de Trabalho

 <p><i>Figura 1 Nome do aluno 1</i></p>	<p>Número : 22350 Nome : Vasco Silvério Curso : Engenharia Informática Turma : B Email : aluno22350@ipt.pt</p>
 <p><i>Figura 2 Nome do aluno 2</i></p>	<p>Número : 16995 Nome : Rúben Garcia Curso : Engenharia Informática Turma : B Email : aluno16995@ipt.pt</p>

Declaração:

Os alunos declaram sob compromisso de honra que o projeto descrito neste relatório é original e da sua autoria com exceção dos seguintes elementos:

Descrição	Fonte
Código do professor foi usado e modificado no projeto.	Classes: Block,Blockchain,BlockchainException, Transactions, Transfer, Histogram, LastBlock,MinerP2P,MiningListener, ServerMiner, AutomaticP2P,RemoteInterface, RemoteObject e todas as classes na biblioteca utils.
Biblioteca JFreeChart	Código open source desenvolvido pela Object Refinery, utilizado na classe pieChart.

2 Introdução

Este relatório visa esclarecer o objetivo do desenvolvimento do projeto final da cadeira de Computação Distribuída, cuja proposta atribuída foi a de desenvolvimento de uma aplicação que faça o uso da tecnologia BlockChain para implementar um sistema de votação eletrónico robusto, seguro, fiável e descentralizado.

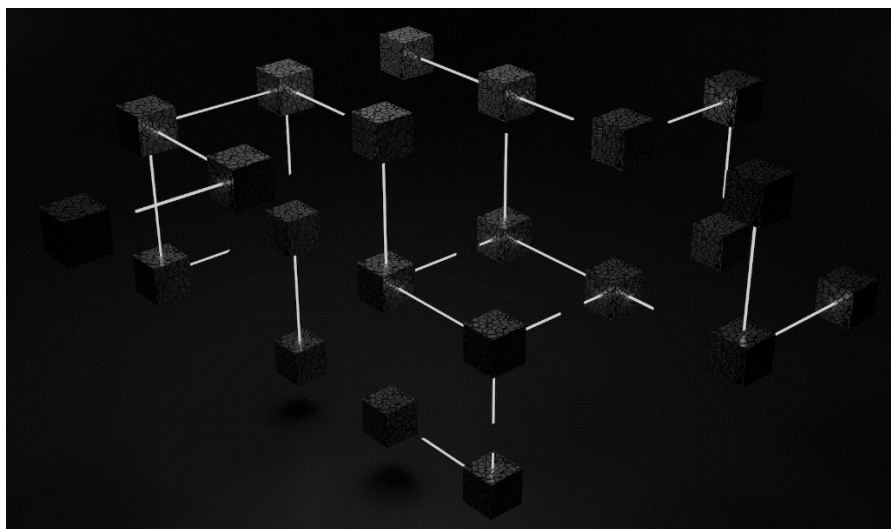


Figura 3 – Blockchain [w1]

2.1 Estado da arte

A *blockchain* define uma estrutura de dados descentralizada e imutável onde cada nó, ou bloco, desta estrutura mantém uma ligação criptográfica ao bloco anterior e contém dados representados criptograficamente, permitindo a verificação confiável de qualquer alteração na cadeia da estrutura, e no conteúdo de cada bloco.

Esta estrutura de dados como é hoje conhecida é o culminar de vários desenvolvimentos criptográficos, matemáticos e informáticos.

No ano 2008 foi inventada uma aplicação de *blockchain* que se popularizou, a *bitcoin*, esta tecnologia propunha usar a *blockchain* como um notário descentralizado para transferências transparentes entre utilizadores representados e verificados por assinaturas, numa rede mantida consensualmente pela maioria que detém o maior poder computacional para validar cada bloco de transações e a totalidade dos dados.

Esta tecnologia porém não resulta de um vácuo, como é explicado no próprio whitepaper da *bitcoin*, os três grandes impulsionadores da *bitcoin* foram o

algoritmo de criptografia *SHA-256*, o mecanismo anti-spam proof-of-work *hashcash*, como também as redes *peer to peer*.

1970	James Ellis, public-key cryptography discovered at Government Communications Headquarters (GCHQ) in secret
1973	Clifford Cocks, RSA cryptosystem discovered at GCHQ in secret
1974	Ralph Merkle, cryptographic puzzles (paper published in 1978)
1976	Diffie and Hellman, public-key cryptography discovered at Stanford
1977	Rivest, Shamir, and Adleman, RSA cryptosystem invented at the Massachusetts Institute of Technology
1979	David Chaum, vaults and secret sharing (dissertation in 1982)
1982	Lamport, Shostak, and Pease, Byzantine Generals Problem
1992	Dwork and Naor, combating junk mail
2002	Adam Bach, Hashcash
2008	Satoshi Nakamoto, Bitcoin
2017	Wright and Savanah, nChain European patent application (issued in 2018)

Figura 4 Algumas das principais descobertas na timeline da blockchain.

Na figura 4 podemos ver algumas das tecnologias que impulsionaram a bitcoin.

O que é a blockchain?

A blockchain armazena os dados em blocos, e relaciona os novos blocos com os anteriores, formando uma corrente. Os dados de cada bloco não podem ser alterados, o que implica que após a escrita dos dados, eles nunca mais vão poder ser editados ou removidos.

A blockchain é um sistema descentralizado, em que o armazenamento dos dados não depende apenas de uma entidade, mas é distribuído por toda a rede que faz parte da blockchain. A descentralização permite distribuir a carga computacional e melhorar a disponibilidade do serviço, contra os sistemas de armazenamento de dados tradicionais que ficam dependentes de servidores únicos.

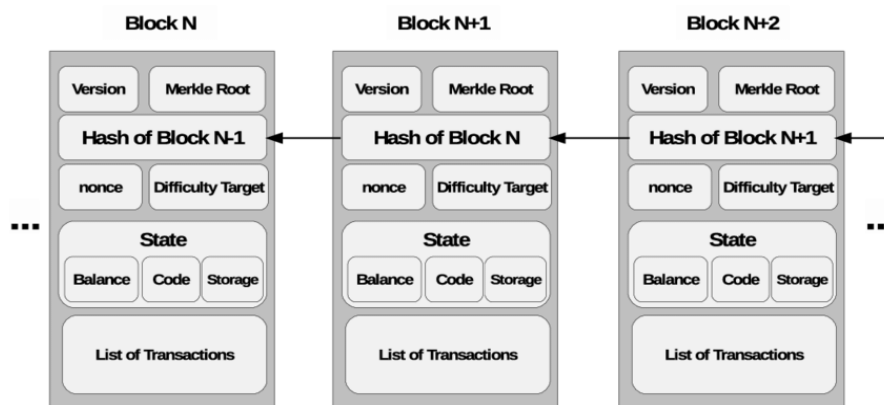


Figura 5 Header de um bloco da blockchain com todos os campos básicos.

O que é um bloco?

Representado na figura 5 está o *header* de um bloco, todos os dados necessários para manter esta estrutura de dados estão aqui presentes:

- Version

Mesmo não sendo um campo obrigatório, é representativo da versão do protocolo que está a ser utilizado pelos miners e nodes da rede.

- Merkle Root

Campo obrigatório no bloco, a merkle root é a representação criptográfica dos dados que vão ser colocados neste bloco, isto é alcançado por uma árvore binária de hash's representativo de cada dado introduzido, sejam representações de factos, imagens ou registos.

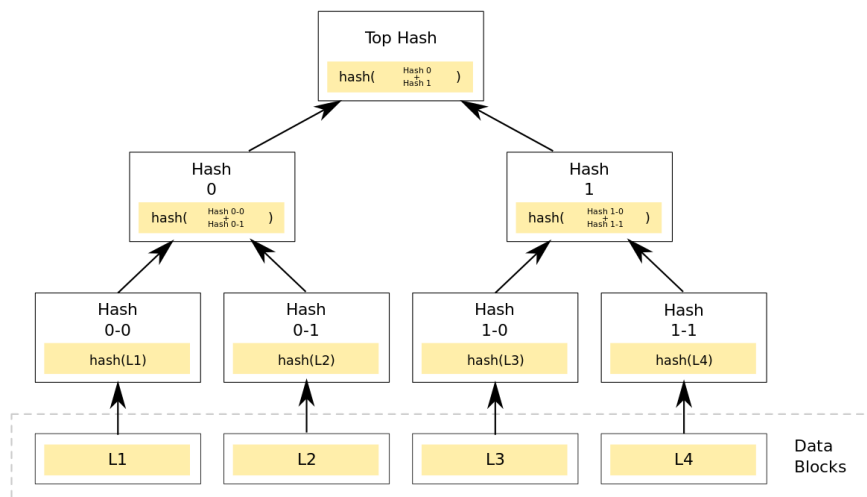


Figura 6 Representação de uma Merkle Tree

Tal como apresentado na figura 6, temos em baixo quatro folhas(L1,L2,L3 e L4) que representam dados introduzidos árvore, estes dados são transformados em hash's que se concatenam pela árvore acima chegando à root que representa todos os dados, se algo se alterar numa folha, deixa de ser válido para a hash obtida na root.

- Hash of Block N

É o hash obtido no bloco anterior, este hash é utilizado para calcular o hash que será adicionado ao bloco seguinte.

- Nonce

Número da solução que resulta do proof-of-work, este número quando concatenado ao hash do bloco anterior e à merkle root do bloco atual, e de seguido representado por hash, apresenta um número determinado de zeros, esse número de zero é definido e quanto maior for mais difícil, moroso e dispendioso é a procura por este número, o que mais uma vez, só é possível pela maioria numa rede.

- Difficulty Target

O número de zeros que se pretende para a solução do bloco, este número pode ser fixo mas deve ser alterado dinamicamente consoante a rede expande e há mais poder computacional para equilibrar a velocidade da adição de blocos e por consequência a estabilidade da blockchain.

Elementos adicionais

- State

Refere-se à condição atual do sistema blockchain, incluindo o status de todas as contas ou contratos inteligentes (instruções computacionais representadas na blockchain). Representa o resultado acumulado de todas as transações anteriores, refletindo o estado atual da rede após a execução de todas as operações.

- List of Transactions / Data

Registo de todas as transações incluídas no bloco. Cada transação contém informações sobre a transferência de ativos, contratos inteligentes ou outras operações executadas pelos participantes da rede. As transações podem ser mantidas numa base de dados tradicional e cruzadas com os hash's da blockchain, mas para aplicações mais sofisticadas, como os contratos inteligentes, é requerido que seja adicionado ao bloco a lógica computacional que permite elaborar processos dinâmicos sobre os dados contidos na blockchain.

O que é proof of work?

Proof of Work (PoW) é um protocolo de consenso utilizado em redes blockchain para validar e processar transações. No PoW, os participantes, chamados de mineradores, competem para resolver problemas computacionais complexos, com já referido, o problema pode ser encontrar o número que adicionado a outro hash devolve um hash com um número determinado de zeros. O primeiro minerador que encontrar a solução correta é autorizado a adicionar um novo bloco à blockchain e é recompensado com novas criptomoedas. É desta forma também que é gerada a moeda na blockchain de moedas digitais.

Quem pode introduzir dados na blockchain?

Os dados são introduzidos na blockchain por meio de transações. Qualquer participante da rede blockchain pode criar uma transação, dependendo das regras e permissões definidas pelo protocolo específico. Uma transação pode incluir a transferência de ativos, execução de contratos inteligentes ou qualquer outra interação suportada pela blockchain. Após a criação da transação, ela é propagada pela rede para ser validada e incluída em um bloco pelos mineradores. Na execução da transação é verificada a assinatura e registada na blockchain a chave pública assimétrica do remetente da transação.

A assinatura de chaves assimétricas em transações é um método de segurança em blockchain que utiliza um par de chaves: uma chave privada, conhecida apenas pelo proprietário, e uma chave pública, divulgada publicamente. Para realizar uma transação, o remetente usa sua chave privada para assinar digitalmente os dados da transação. A assinatura é verificada pela chave pública correspondente, que garante a autenticidade e a integridade da transação. Essa abordagem fornece segurança, pois apenas o detentor da chave privada pode gerar uma assinatura válida, protegendo as transações contra adulteração e garantindo a identidade do remetente.

O que é um miner?

Um miner, ou minerador, é um participante na rede blockchain que se dedica à mineração, o processo de encontrar a solução para um problema criptográfico complexo, como já referido. Os mineradores competem para validar transações e adicionar novos blocos à blockchain. Além da recompensa pela mineração bem-sucedida (geração de novas criptomoedas), os mineradores podem receber taxas de transação pelos usuários que enviam transações.

O que é um node?

Um nó, ou node, é um dispositivo que faz parte da rede blockchain. Cada nó tem uma cópia do livro-razão, ou ledger distribuído (a blockchain) e participa no processo de validação e consenso. Existem dois tipos principais de nós:

nós completos (full nodes) e nós leves (light nodes). Os nós completos mantêm uma cópia completa do histórico da blockchain e participam ativamente na validação. Os nós leves, por outro lado, confiam nos nós completos para obter informações, sendo mais leves em termos de requisitos de armazenamento e processamento. Todos os nós têm a capacidade de propagar transações e blocos pela rede.

Proof of Work vs Proof of Stake

É importante referir que o proof of work tem as suas limitações, a despesa de energia que é necessária tal como a vulnerabilidade a uma maioria na rede que seja centralizada é um dos exemplos.

Existem outras soluções de protocolo para a blockchain, um delas é o proof of stake.

Para validar um bloco no protocolo proof of stake basta introduzir um assinatura num bloco e adicioná-lo à blockchain, porém isto requer um passo inicial onde um validador eleito necessita de colocar tokens que estão previamente disponibilizados por ele, ou seja, numa pool de validação e que são obtidos anteriormente na cadeia para a sua chave pública.

Se esta transação for inválida, o validador perde os seus tokens e é garantido que mesmo que este tenha corrompido os factos para obter tokens, este irá sempre perder mais tokens do que irá receber na corrupção dos fatos do ledger.

Esta solução apresenta assim um meio mais escalável para uma blockchain que pretende ser mais rápida e mais eficiente.

Esta solução é implementada na blockchain Ethereum, que oferece uma Virtual Machine descentralizada e distribuída, mantida por uma blockchain que contém o código que é validado e processado pela rede de mineiros e nodes em instruções básicas na linguagem Solidity. Para poder fazer aplicações robustas e eficientes, sejam websites, alojamento ou aplicações é necessário que esta rede blockchain seja rápida no processamento destas intruções, isto levou a melhoramentos neste protocolo que visaram criaram layers mais eficientes sobre esta virtual machine, a EVM(Ethereum Virtual Machine), exemplos destes protocolos são o Arbitrum ou o Optimism, que facilitam as validações do código solidity e simplificam a validação dos blocos para os nodes da rede mantendo a confiabilidade da rede de layer 1.

2.2 Exemplos de aplicações e projetos baseados em blockchain

- **Moedas Digitais:**
 - **Bitcoin** – A bitcoin foi a moeda digital que veio revolucionar o mundo, e foi o projecto que mais impacto teve a divulgar a tecnologia blockchain. Esta moeda digital aparece no ano de 2008.
 - **Ethereum** – A Ethereum aparece no ano de 2015, e de forma similar à Bitcoin, também usa a blockchain para armazenar os dados.
- **Sistemas financeiros:**
 - **Barclays** – A Barclays tem a visão de que a blockchain vai aproximar as instituições financeiras e facilitar a colaboração global. A segunda visão é de que vai criar eficiências no processamento dos dados financeiros.
- **Consultoria informática:**
 - **GardTime** – A GuardTime disponibiliza aplicações para empresas baseadas em blockchain, em destaque são aplicações de sector público, de saúde, construção, logística, energia e sistemas financeiros.
 - **Remme** – Tal como a GardTime, a Remme também disponibiliza para empresas aplicações baseadas em Blockchain.
- **Saúde:**
 - **SimplyVital Health** – A SimplyVital Health desenvolveu duas aplicações baseadas em blockchain: a ConnectingCare, que é uma aplicação destinada ao uso por hospitais, e a Health Nexus, que pode ser usada por qualquer pessoa para trocar informações sobre o sistema de saúde.
 - **MedRec** – Com o desenvolvimento pelo MIT, o MedRec é uma plataforma baseada em blockchain que permite registar dados dos doentes que permite o acesso a esses dados por diferentes organizações.
- **Votações:**
 - **Voatz** – É uma aplicação blockchain para votações eletrônicas. Utiliza a tecnologia blockchain para garantir a segurança, integridade e transparência do processo de votação. Eleitores podem participar em eleições usando dispositivos móveis, e os resultados são registados de forma imutável na blockchain.
 - **Horizon State** – Esta plataforma utiliza blockchain para possibilitar votações seguras e transparentes. Ao empregar contratos inteligentes, a Horizon State elimina riscos de manipulação,

proporcionando uma solução eficaz para processos democráticos eletrônicos.

- **Oracles:**

- **Chainlink** – Chainlink é uma plataforma de oracles descentralizada. que conecta contratos inteligentes com fontes de dados do mundo real, permitindo que informações externas, como preços de ativos, sejam incorporadas aos contratos. Isso expande a utilidade dos contratos inteligentes para casos que exigem dados externos.
- **VeChain** – VeChain também atua como uma plataforma de oracles para integrar informações do mundo físico à blockchain. Isso é especialmente útil em setores como cadeia de suprimentos, onde dados em tempo real são cruciais.

- **Smart Contracts:**

- **Ethereum** – Permite que desenvolvedores criem contratos autoexecutáveis, sem a necessidade de intermediários. Isso abre um vasto leque de aplicações, desde finanças descentralizadas (DeFi) até jogos baseados em blockchain.
- **Cardano** – É uma plataforma de contratos inteligentes que se destaca pela abordagem científica e foco em segurança. Procura oferecer uma infraestrutura eficiente e segura para contratos inteligentes, com aplicações em diversas áreas.

- **Armazenamento de Dados:**

- **Filecoin** – Filecoin utiliza blockchain para criar um mercado descentralizado de armazenamento. Utilizadores podem alugar espaço de armazenamento de outros participantes na rede, e a blockchain registra e valida essas transações, garantindo a integridade do sistema.

3 Descrição do projeto

O projecto ***Election Ledger*** é um sistema de votação eletrónico, que usa a tecnologia Blockchain para o armazenamento de dados de forma descentralizada. Este projeto permite que o responsável pela Mesa Eleitoral carregue os candidatos e os eleitores, e inicie uma votação pública.

É usado o conceito de fases ao longo do decorrer da eleição. A fase 0 é usada para criar o bloco inicial, a fase 1 é usada para criar os candidatos da eleição, a fase 2 é usada para criar os eleitores, a fase 3 é usada para que seja permitida a votação, e a fase 4 é onde são disponibilizados os resultados da eleição.

São usados sistemas de segurança através de chaves assimétricas, que garantem a privacidade dos dados, e assinaturas das transações para garantir o não repúdio.

O processamento deste projecto foi desenhado para ser distribuído por vários computadores, o que permite distribuir a carga de computação, aliado à descentralização do armazenamento dos dados da blockchain.

O conceito deste projeto não é novidade, e importa referir que existem projetos similares usados por outras instituições, como por exemplo na Austrália, onde é usado o sistema *Vote Australia*.

4 Trabalho 1 - Blockchain

4.1 Arquitetura da aplicação



Figura 7 - Diagrama de classes do package Blockchain

O Package *electionledger.blockchain* usa 3 classes, denominadas:

- **Block** – Esta classe é usada para construir um objeto do tipo bloco;
- **Blockchain** – Esta é a classe usada para criar o objeto *BlockChain*. Esta classe usa vários objetos da classe *Block*.
- **BlockchainException** – Esta classe é usada pelas outras duas classes pertencentes a este Package, e serve para gerar excessões em ambas.

4.2 Biblioteca electionledger.blockchain

- A *electionledger.blockchain* é usada para fazer a gestão da *BlockChain*, incluindo as várias verificações necessárias para o seu correto funcionamento. Também é aqui que são criados objetos do tipo *Block* que por sua vez, caso as validações sejam efetuadas com sucesso, é adicionado ao fim da *BlockChain*

4.2.1 Classe Block.java

Esta classe permite criar objetos do tipo *Block*, e irá representar todos os blocos da blockchain. As validações do bloco são feitas nesta classe.

Esta classe implementa o *Serializable*, porque se tratar de uma classe que tem de ser sincronizada com outros servidores através de RMI.

Uso de classes:

- MinerP2P – Esta classe é usada para obter o Nonce que terá sido descoberto por um dos mineiros.
- Hash – Esta classe é usada para converter o Nonce do novo bloco, a Hash do bloco anterior e o timestamp para uma Hash.
- MerkleTree – Esta classe é usada para obter a raiz da árvore de Merkle com os dados do novo bloco

Atributos:

Strings:

- previousHash – Refere a hash do bloco anterior
- merkleTreeRoot – Refere a root da árvore de Merkle com os dados fornecidos
- currentHash – Refere a Hash do bloco atual
- data – Dados do bloco atual

Inteiros:

- DIFICULTY – Dificuldade para minar o bloco. Por defeito é 3. Este é um atributo público e estático para que possa ser definido por outras classes.
- nonce – Refere a nonce do bloco atual
- phase – Refere a fase do bloco atual
- numberOfZeros – É igualada à DIFICULTY e serve para determinar com quantos zeros a Hash deve começar. Por defeito, é iniciado a 3.

Longos:

- timestamp – Este atributo armazena o tempo de quando é que o bloco foi minado

Métodos:

- getTransactions() – Serve para devolver as transações de um bloco. Retorna um CopyOnWriteArrayList de Strings, onde cada String é uma transação do bloco.
- getHeader() – Serve para devolver o cabeçalho do bloco. Retorna uma string.
- getInfo() – Serve para devolver várias informações do bloco, como a Hash do bloco anterior, os dados do bloco atual, o seu Nonce, a sua Hash, o número de zeros, a root da árvore de Merkle, a fase, e se o bloco é válido. Retorna uma string.
- setNonce(int nonce) – Recebe como parâmetro um inteiro que será atribuído ao nonce do bloco a minar. Também é neste método que é calculada a Hash do bloco.
- getNumberOfZeros() – Retorna como inteiro a quantidade de zeros da Hash do bloco.
- getMiningData() – Retorna como string os dados a minar.
- getPreviousHash() – Retorna como string a Hash do bloco anterior.
- getData() – Retorna os dados do bloco em forma de string.
- getPhase() – Retorna a fase da eleição em que o bloco foi minado.
- getMerkleTreeRoot() – Retorna em string a root da árvore de Merkle do bloco.
- setMerkleTreeRoot(String merkleTreeRoot) – Recebe como parâmetro uma string com a root da Merkle Tree, e atribui esse valor à root da Merkle Tree do bloco.
- getNonce() – Retorna como inteiro o nonce do bloco.
- getCurrentHash() – Retorna a hash do bloco.
- getTimeStamp() – Retorna o timestamp de quando o bloco foi minado.
- getSerialVersionUID() – Retorna o serialVersionUID.
- calculateHash() – Retorna como string o calculo da nova hash.
- toString() – Retorna como string os dados do bloco.
- isValid() – Verifica se o bloco é válido. Retorna um booleano *true* ou *false*.

4.2.2 Classe Blockchain.java

Esta classe permite criar a Blockchain, onde serão adicionados os Blocos, e implementa o *Serializable*, porque se tratar de uma classe que tem de ser sincronizada com outros servidores através de RMI.

Uso de classes:

- Block – Esta classe é usada para construir o primeiro bloco da Blockchain.
- MerkleTree – Esta classe é usada para obter a raiz da árvore de Merkle para o primeiro bloco da Blockchain.

Atributos:

CopyOnWriteArrayList:

- Chain – É o CopyOnWriteArrayList que contém todos os blocos da blockchain.
- genesis – CopyOnWriteArrayList usado para poder adicionar o primeiro bloco à blockchain.

Métodos:

- getLastBlockHash() – Serve para devolver a hash do último bloco da blockchain. Retorna uma string.
- getLastBlock() – Serve para devolver o último bloco da blockchain. Retorna um *Block*.
- addBlock(Block newBlock) – Recebe com parâmetro um *Block* e serve para adicionar um novo bloco ao fim da blockchain. Só permite adicionar o bloco se for válido.
- addVerifiedBlock(Block block) - Recebe com parâmetro um *Block* e serve para adicionar um novo bloco ao fim da blockchain que já passou nas validações.
- getChain() – Retorna como CopyOnWriteArrayList a blockchain.
- setChain(CopyOnWriteArrayList<Block> chain) – Recebe como parâmetro um CopyOnWriteArrayList e serve para inicializar uma nova blockchain.
- getBlock(int index) – Recebe como parâmetro um inteiro e devolve um *Block* com o índice recebido.
- toString() – Devolve os blocos da blockchain em String.

- `load(String fileName)` – Recebe com parâmetro uma string que representa o nome do ficheiro da blockchain que vai ser carregado.
- `isValid()` – Verifica se um bloco é válido. Retorna um booleano.

4.2.3 Classe `BlockchainException.java`

Esta classe serve para ajustar as exceções da blockchain. Herda a class *Exception* para poder gerar os erros.

Métodos:

- `BlockchainException(String msg)` – Recebe como parâmetro uma string que representa o erro a ser mostrado ao utilizador.
- `show()` – Serve para mostrar um `JOptionPane` com o erro ao utilizador.
- `printStackTrace()` – Imprime no log o erro da exceção.

4.2.4 Classe `Transactions.java`

A classe `Transactions` armazena as transações recebidas pelos clientes e sincronizadas por outros mineradores.

Uso de classes:

Esta classe é acedida pela `RemoteObject` e pela `Block` para obter as transações e minerar o novo bloco com essas transações.

Atributos:

Inteiros:

- `MAXTRANSACTIONS` – Este atributo define a quantidade máxima de transações por bloco.

`CopyOnWriteArrayList`

- `List` – Este atributo armazena as transações recebidas.

Métodos:

- **getList()** – Este método retorna a lista de transações em forma de `CopyOnWriteArrayList`.
- **contains(String trans)** – Este método recebe como parâmetro uma string que é uma transação e verifica se essa transação já existe. Retorna um booleano.
- **addTransaction(String newTrans)** – Este método recebe como parâmetro uma string que é uma nova transação, e adiciona essa transação à lista de transações.
- **removeTransactions(CopyOnWriteArrayList<String> lst)** – Este método recebe como parâmetro um `CopyOnWriteArrayList` de transações e remove as transações atuais que coincidem com as recebidas por esse parâmetro.
- **synchronize(CopyOnWriteArrayList<String> other)** – Este método recebe como parâmetro um `CopyOnWriteArrayList` e serve para adicionar transações recebidas às transações atuais.

4.2.5 Classe Transfer.java

A classe transfer serve para criar transferências entre um remetente, destinatário, com um certo valor e uma assinatura.

Uso de classes:

Esta classe é usada na `RemoteObject` para criar transferências com os dados recebidos dos clientes ou quando se faz a leitura de transações de blocos já existentes. Após a criação de uma transferência, ela é convertida para uma Hash em forma de string e armazenada na classe `Transactions`.

Atributos:**String:**

- **From** – Serve para identificar qual é o utilizador de origem da transferência.
- **To** – Serve para identificar qual é o destinatário da transferência.
- **Signature** – Assinatura com a chave privada de quem efetuou o voto.

Métodos:

- **getValue()** – Retorna em formato de inteiro o valor da transferência.
- **getSignature()** – Retorna em formato de string a assinatura da transferência.
- **setSignature(String signature)** – Recebe como parâmetro uma string referente à assinatura da transferência e atribui esse valor ao atributo signature.
- **setValue(int value)** – Recebe como parâmetro um inteiro referente ao valor da transferência e atribui esse valor ao atributo value.
- **getFrom()** – Retorna em formato de String o utilizador de origem da transferência.
- **setFrom(String from)** – Recebe como parâmetro uma String e atribui essa String ao atributo From.
- **getTo()** – Retorna em formato de String o destinatário da transferência.
- **setTo(String to)** – Recebe como parâmetro uma String referente ao destinatário da transferência e atribui essa String ao atributo To.
- **toString()** – Retorna em formato de String os detalhes da transferência: A origem, o destinatário e o valor.
- **toText()** – Retorna em formato de String a conversão do objeto Transfer para Base64.
- **fromText(String obj)** – Recebe como parâmetro uma String respeitante a um objeto Transfer convertido para Base64, e devolve novamente essa String em um objeto Transfer.
- **hashCode()** – Retorna em inteiro o código da hash.
- **Equals(Object t)** – Recebe como parâmetro um objeto e compara com a transferência. Retorna um booleano a indicar se são o mesmo objeto ou não.

5 Trabalho 2 – Computação Multitarefa

5.1 Arquitetura da aplicação

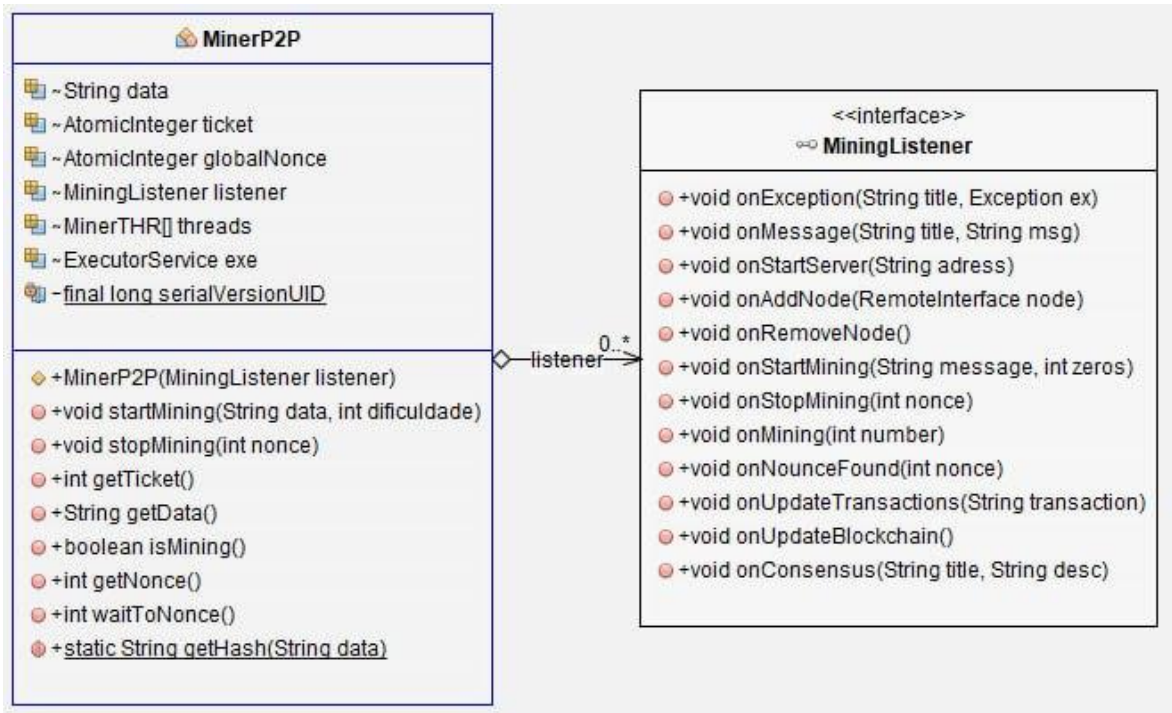


Figura 8 - Diagrama de classes do package Miner

O Package *electionledger.miner* usa 2 classes, denominadas:

- **MinerP2P** – Esta classe é usada para a mineração, cujo objetivo é o de achar a nonce de um bloco com recurso a multithreads.
- **Mining Listener** – Esta interface é usada como listener do mineiro.

Biblioteca electionledger.miner

Na biblioteca *electionledger.miner* estão as classes necessárias para criar um mineiro e para o seu correto funcionamento.

5.1.1 Classe MinerP2P.java

Uso de classes:

A interface `MiningListener` é usada e referenciada como “listener” para identificar quando é que o mineiro se encontra a minar ou se está parado.

A classe `MinerTHR` é uma “Inner Class” e é usada pelo `MinerP2P` para usar o multithreading.

Atributos:

String:

- `Data` – Este atributo armazena os dados para a mineração.

AtomicInteger:

- `Ticket` – É usado para que cada mineiro retire um ticket e incremente até que o nonce seja descoberto.
- `globalNonce` – É atribuído o nonce, no fim de descoberto por um mineiro, a este atributo.

MiningListener:

- `Listener` – Referência à interface `MiningListener` para verificar quando é que o mineiro está a minar ou está parado.

MinerTHR:

- `Threads` – Array com as threads que vão executar a Inner Class `MinerTHR`.

ExecutorService:

- `Exe` – Executor usado para a execução das threads e controlar a sua paragem.

Métodos:

- **`startMining(String data, int dificuldade)`**: Recebe como parâmetros em formato de String os dados e como Inteiro a dificuldade do bloco a minar. Obtém o número de threads disponíveis no sistema, e usa o Executor para executar a classe

MinerTHR com cada thread para que seja minerado um novo bloco.

- **stopMining(int nonce):** Recebe como parâmetro um inteiro que refere a nonce descoberta por um mineiro. Atribui esse nonce à variável globalNonce para que outros mineiros de outras threads saibam que o nonce foi descoberto e parem a mineração.
- **getTicket():** Obtém o próximo ticket para testar a mineração. Retorna em formato de Inteiro.
- **getData():** Devolve como retorno uma String com os dados da mineração.
- **isMining():** Retorna como booleano a validação se o mineiro está a minar ou está parado.
- **getNonce():** Retorna como inteiro o nonce descoberto.
- **waitToNonce():** Retorna como Inteiro a validação do Executor que fica a aguardar pela descoberta do nonce para terminar as threads.

5.1.2 Classe Interior MiningTHR

Uso de classes:

A classe MinerP2P usa esta classe para a descoberta do nonce, com todos os threads disponíveis do sistema.

Atributos:

AtomicInteger:

- myTicket – Ticket usado pela thread.
- myNonce – Nonce usado pela thread.

Integer:

- difficulty – Dificuldade do bloco a minerar.

String:

- myData – Dados do bloco a minerar.

MiningListener:

- Listener – Usado para descobrir se o mineiro que está a usar essa thread está a ser usado ou se está parado.

Métodos:

- **stop(int number):** Recebe como parâmetro um inteiro referente ao nonce descoberto por um mineiro e atribui ao myNonce esse valor. Posteriormente manda parar essa thread.
- **Run():** método para executar a thread. É usado um ciclo *while* que verifica quando é que o nonce é descoberto. Quando for descoberto, manda parar as outras threads ao chamar o onNounceFound do listener.
- **getHash(String data):** Recebe como parâmetro uma String referente aos dados para converter para hash, e retorna a hash desses dados.

5.1.3 Classe MiningListener.java

É uma interface usada por *listeners* que verifica condições como a descoberta de um nonce, a adição de um novo mineiro, entre outras, para que sejam executadas acções que ficam à “escuta”.

Uso de classes:

O MiningListener é usado na classe MinerP2P para que quando uma nova thread seja lançada, seja enviado como parâmetro um *listener* que verifica quando é que um nonce é descoberto.

Também a classe RemoveObject usa o *listener* para validar quando existe a adição de um novo mineiro à rede, a remoção de um mineiro, ou uma atualização na blockchain.

6 Trabalho 3 – Computação Distribuída

6.1 Arquitetura da aplicação

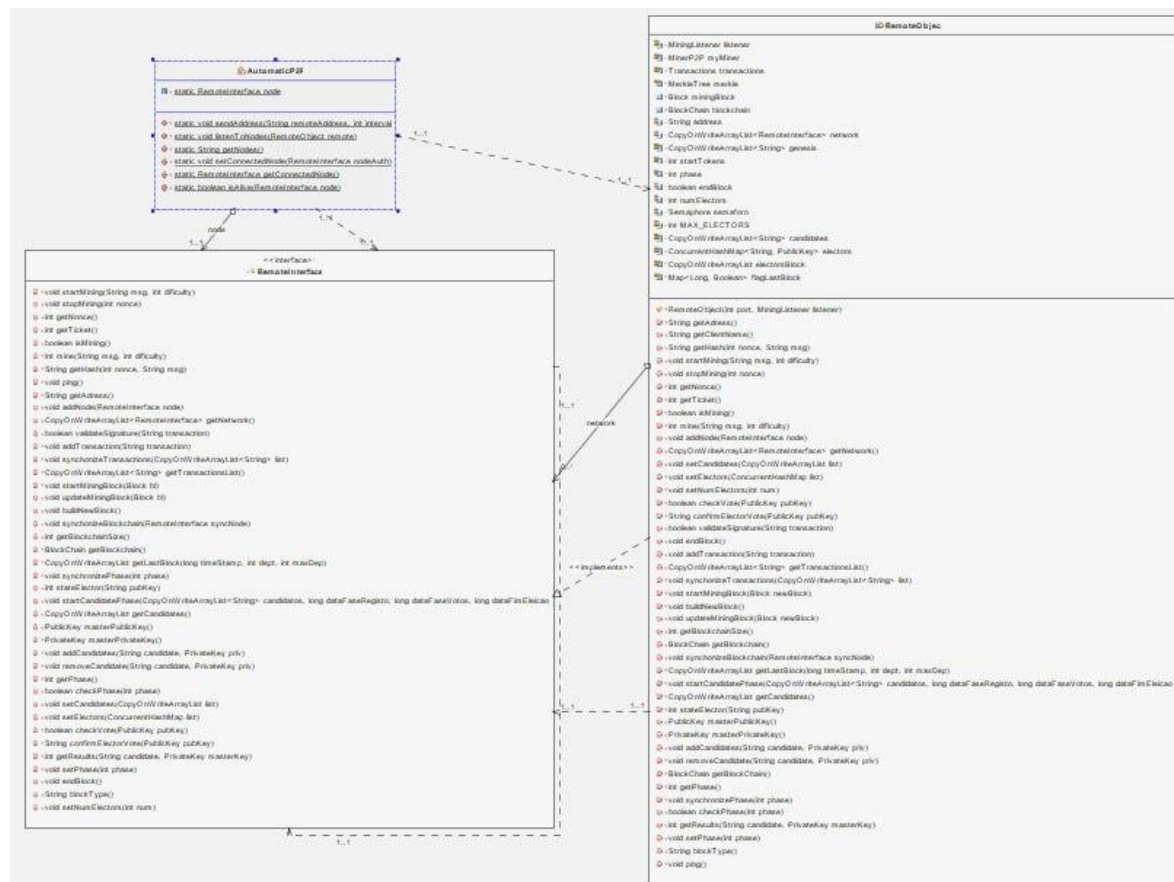


Figura 9 - Diagrama de classes do package node

O Package *electionledger.node* usa 2 classes, denominadas:

- **AutomaticP2P** – Usa o protocolo UDP para a descoberta de novos nós que se estejam a ligar à rede. Nesta classe também verifica-se se um nó deixou de responder para que seja possível o remover da rede.
- **RemoteInterface** – Interface usada pelo RemoteObject para implementar todos os métodos necessários para a sincronização entre todos os nós da rede.
- **RemoteObject** – Classe que implementa a interface RemoteObject e que vai ser usada por todos os nós da rede e trata de os manter sincronizados.

Biblioteca electionledger.node

Na biblioteca electionledger.node estão as classes *core* da aplicação, onde se encontra a classe referente aos objetos remotos e que trata de mandar sincronizar toda a informação entre todos os nós da rede. Estas classes são como o orquestrador da aplicação, que instruem os mineiros a iniciar uma mineração enviado os dados para o novo bloco que devem minerar, se devem de parar de minerar, geram a rede de mineiros adicionando e removendo novos nós e adicionam novos blocos à blockchain.

6.1.1 Classe AutomaticP2P.java

Uso de classes:

Usa a RemoteInterface para atribuir de forma estática a ligação que está a ser feita entre um cliente e os nós a que se está a ligar. A classe ServerMiner também usa esta classe para ficar a escutar quando é que existem pacotes UDP recebidos, obter os endereços dos remententes desses pacotes UDP, e ligar-se automaticamente a esses remetentes. Também a classe serverMiner usa esta classe para verificar quando é que um nó deixou de responder e remover esse nó da sua rede de mineiros.

Atributos:

RemoteInterface:

- **Node** – Armazena de forma estática o endereço do nó ao qual o cliente se está a ligar.

Métodos:

- **sendAddress(String remoteAddress, int interval):** Recebe como parâmetro uma String com o endereço de destino e um Integer que define o tempo de intervalo do envio de pacotes UDP para a deteção automática de terceiros de que este nó se encontra ativo e pronto para estabelecer conexões.
- **listenToNodes(RemoveObject remote):** Recebe como parâmetro um RemoteObject e fica à escuta da receção de pacotes UDP enviados por outros nós. Caso receba pacotes, adiciona o nó que enviou esses pacotes à rede de mineiros.

- **getNodes():** Métodos estático usado pelos clientes que obtém os nós que estão na rede e retorna o primeiro que receber em forma de String.
- **setConnectedNode(RemoteInterface nodeAuth):** Método estático que atribui ao atributo “node” o nó que o cliente vai usar para estabelecer ligação com a rede.
- **getConnectedNode():** Retorna o objecto RemoteInterface com a referência de a que nó o cliente está ligado.
- **isAlive(RemoteInterface node):** Retorna um booleano que testa a ligação a um nó com recurso ao método “ping” na classe RemoteObject.

6.1.2 Classe RemoteInterface.java

É uma interface usada por *mineiros* que é implementada na RemoteObject para executar as ações em objetos remotos e sincronizar os objetos remotos. Esta interface implementa métodos relacionados com o mineiro, com a rede de mineiros, com as transações efetuadas, com os blocos, com a blockchain, métodos de consenso e métodos referentes à eleição, descritos com maior detalhe na referência abaixo, na classe RemoteObject.

6.1.3 Classe RemoteObject .java

Herda a classe UnicastRemoteObject e implementa a interface RemoteInterface. Esta classe é considerada o core da aplicação por ser responsável por praticamente todas as funções deste sistema. E

Uso de classes:

Usa a classe MinerListener para criar um objeto *listener* que fica à escuta de ações realizadas e despoleta a sincronização entre todos os nós quando deteta uma dessas ações. Entre elas: adições de novos nós na rede, remoção de nós na rede, início de mineração, adições de transações ou atualizações na blockchain.

Usa a classe MinerP2P para dar ordem de mineração quando são coletadas as transações necessárias para construir um novo bloco.

Usa a classe Transactions para registar as transações pendentes de mineração.

Usa a classe MerkleTree para obter a raiz da árvore de Merkle com os dados do bloco que vai ser minerado.

Usa a classe Block para criar novos blocos para serem minerados.

Usa a class Blockchain para criar e referenciar a blockchain que está em uso.

Atributos:

MiningListener:

- Listener – usado para escutar eventos e atuar quando existe a adição ou remoção de um novo nó à rede, quando existe a adição de uma nova transação, ou quando existe uma atualização na blockchain.

MinerP2P:

- myMiner – Referencia o mineiro para o executar quando for necessário minerar um novo bloco.

Transactions:

- Transactions – Referencia as transações que estão pendentes de mineração e que vão ser usadas para minerar o próximo bloco.

MerkleTree:

- Merkle – Construi uma merkle tree com os dados para que seja gerada a raiz da árvore de Merkle para o novo bloco que vai ser minerado.

Block:

- miningBlock – Referencia o novo bloco que vai ou está a ser minerado.

Blockchain:

- Blockchain – Referência para a blockchain que está a ser usada no sistema.

String:

- Address – Guarda o endereço no nó.

CopyOnWriteArrayList:

- Network – Armazena todos os nós da rede onde se está a conectar.
- Genesis – Auxiliar para criar o primeiro bloco da blockchain.
- Candidates – Obtém os candidatos da eleição para posteriormente minerar um bloco com eles.
- Electors – Obtém os eleitores que podem votar na eleição para posteriormente minerar um bloco com eles.

Integer:

- startTokens - Tokens usados para o início da eleição.
- Phase – Fase em que se encontra a eleição. Existem 4 estados possíveis:
 - 0 – Início da eleição
 - 1 – Registo de candidatos
 - 2 – Registo de eleitores
 - 3 – Votação por parte dos eleitores
 - 4 – Término da eleição, divulgação da chave privada da mesa eleitoral, e disponibilização de resultados.
- numElectors – Número de eleitores.

Métodos:

- **getAddress():** Retorna em formato de String o endereço do nó corrente.
- **getClientName():** Retorna em formato de String o nome do nó corrente.
- **getHash:** Recebe como parâmetros um inteiro com o nonce do novo bloco minerado, e como String os dados do bloco. Retorna em formato de String a Hash gerada para o bloco.
- **startMining(String msg, int difficulty):** Recebe como parâmetros uma String com os dados do bloco a ser minerado e como Integer a

dificuldade do bloco a ser minerado. Usa o MinerP2P para minerar esse bloco.

- **stopMining(int nonce):** Recebe como parâmetro um integer com o nonce descoberto pelo nó. Caso o mineiro esteja a trabalhar, manda-o parar e distribui essa informação pela rede.
- **getNonce():** Retorna em Integer o nonce do mineiro.
- **getTicket():** Retorna em Integer o ticket do mineiro.
- **isMining():** Retorna um booleano que indica se o mineiro está a minerar ou está parado.
- **Mine(String msg, int difficulty):** Dá ordem para o mineiro começar a minerar com os dados recebidos e a dificuldade atribuída.
- **addNode(RemoteInterface node):** Recebe um novo nó como parâmetro de entrada, e verifica se não está na sua rede. Se não estiver, adiciona-o à sua rede.
- **getNetwork():** Retorna um CopyOnWriteArrayList com a lista de nós da rede.
- **setCandidates(CopyOnWriteArrayList list):** Recebe como parâmetro um CopyOnWriteArrayList com a lista de candidatos e atribui essa lista ao atributo “candidates”.
- **setElectors(ConcurrentHashMap list):** Recebe como parâmetro um ConcurrentHashMap com a lista de eleitores e atribui essa lista ao atributo “electors”.
- **setNumElectors(int num):** Recebe como parâmetro um Integer e atribui essa variável ao atributo numElectors que indica a quantidade de eleitores que existem na eleição.
- **checkVote(PublicKey pubKey):** Recebe uma chave pública e intera todos os blocos da blockchain à pesquisa de referências de transações que foram feitas com essa chave. Verifica no “To” de cada transação se coincide com a chave pública, e se coincidir, significa que a mesa eleitoral deu um token para essa entidade poder votar. Verifica também no “From” se coincide com a chave pública, e se coincidir, significa que essa entidade já gastou o seu token num voto. Por fim retorna um booleano que indica se a entidade já votou.

- **confirmElectorVote(PublicKey pubKey):** Recebe como parâmetro uma chave pública e itera todos os blocos da blockchain à pesquisa de um voto que tenha sido feito com essa chave pública. Quando e se achar, retorna em formato de String a informação do bloco onde o voto dessa chave pública foi registado, o partido em quem votou descriptado, e a validação se os dados são íntegros com a validação da root da Merkle Tree.
- **validadeSignature(String transaction):** Recebe como parâmetro uma transação e verifica a assinatura dessa transação ao usar a chave pública armazenada no "From". Retorna um booleano que indica se a assinatura da transação é válida ou não.
- **addTransaction(String transaction):** Se ainda não tiver armazenada a transação recebida como parâmetro, verifica a assinatura da transação e se passar nessa validação, adiciona a transação à lista "transactions". Envia para o listener a informação de que recebeu uma transação, e faz a verificação pela qual em que fase da eleição está:
 - **Fase 0:** É minerado um bloco com apenas a transação inicial, do System para o Master, com a quantidade de tokens igual à quantidade de eleitores esperados para a votação.
 - **Fase 1:** É minerado um bloco com todos os candidatos que vão poder ser eleitos durante a votação.
 - **Fase 2:** É minerado um bloco com todos os eleitores que vão poder realizar a votação.
 - **Fase 3:** É iniciado o período de votação. Durante este período são verificadas todas as transações recebidas e a quantidade de transações em espera para serem mineradas. Quando esse valor for igual ou superior a MAXTRANSACTIONS, que por defeito é igual a 2, é minerado um novo bloco. O último bloco a ser minado pode ter menos de MAXTRANSACTIONS, e para forçar a mineração do bloco da fase 3 para posteriormente iniciar a fase 4, é usada uma variável booleana "endBlock" que passa a true quando a mesa eleitoral dá ordem para terminar a eleição caso existam transações pendentes. O bloco é assim minerado e por fim a eleição passa para a fase 4.
 - **Fase 4:** Nesta fase é apenas minerado um bloco com uma única transação, referente à devolução dos tokens do Master para o System.

Em todas as sincronizações de transações solicitadas para os nós da rede, é verificado através do método *isAlive* se o nó ainda se encontra activo. Caso não haja resposta, esse nó é removido da rede.

- **getTransactionsList():** Devolve como CopyOnWriteArrayList a lista de transações.
- **synchronizeTransactions(CopyOnWriteArrayList<String> list):**

Recebe como parâmetro um CopyOnWriteArrayList com uma lista de transações. Para cada transação nessa lista, tenta adicionar a transação às suas transações. Caso já exista, não acontece nada. Por fim, envia para os restantes nós da rede a sua lista de transações. Caso algum nó da rede não responda, é removido da sua rede.

- **StartMiningBlock(Block newBlock):** Recebe como parâmetro o novo bloco que vai ser minerado. Obtém as transações que estão em espera para serem adicionadas, limpa essa lista de transações, sincroniza o bloco com os restantes mineiros que tem na sua rede, e por fim executa o método startMining() para iniciar a mineração do novo bloco.
- **bluidNewBlock():** Antes de iniciar a construção do novo bloco, este método sincroniza a blockchain com todos os nós. Posteriormente, constrói um novo bloco com os dados das transações, com a raiz da árvore de Merkle, com a hash do último bloco, com a dificuldade de mineração, e com a fase em que se encontra a eleição. Por fim, executa a mineração do novo bloco. Se este método for chamado numa altura em que esteja na fase 0, 1, 2 ou 4 e a lista de transações estiver vazia, é feito o return para sair do método. Caso esteja na fase 3, e como nessa fase é permitida a mineração de vários blocos com o número de transações definidas no atributo "MAXTRANSACTIONS", vai ser feito o return quando o número de transações for menor que esse valor para sair do método e não gerar um bloco com menos do que essas transações. Caso seja o último bloco da fase 3, mesmo que esse valor do número de transações seja inferior, o bloco tem de ser minado com as transações que existem e desta forma entra aqui em uso a variável booleana endBlock que caso seja verdadeira, caso não existam transações e caso a eleição esteja na fase 3, é feito o return.
- **updateMiningBlock(Block newBlock):** Recebe como parâmetro um bloco referente ao novo bloco acabado de minerar. Verifica se esse bloco é válido tendo em conta se a sua hash tem de facto os zeros necessários no início da hash e se a hash referente ao bloco anterior coincide. Caso passe nas validações, o novo bloco é

adicionado à blockchain. Posteriormente, é sincronizado com os restantes nós da rede.

- **getBlockchainSize():** Retorna como Integer o tamanho da blockchain.
- **getBlockChain():** Retorna a blockchain.
- **synchronizeBlockChain(RemoteInterface syncNode):** Recebe como parâmetro um nó que vai usar para sincronizar a sua blockchain. Verifica entre os dois nós qual é o que tem mais blocos. Adiciona ao nó que tem menos blocos os restantes blocos detidos pelo outro nó. Por fim, sincroniza a sua blockchain com os restantes nós.
- **getLastBlock(long timeStamp, int dept, int maxDep):** Usa o consenso para determinar qual é que é de facto o último bloco da blockchain. Usa uma flag booleana *flagLastBlock* para determinar se já respondeu ao pedido de outros nós para lhes fornecer qual é o seu último bloco. Se já tiver respondido anteriormente, é feito o return para sair do método. Se não, verifica se o pedido pelo último bloco chegou ao fim do encadeamento de 7 nós. Se for superior, também é feito o return para sair da função, se não, é iniciado um array de threads do tamanho da rede de nós, e para cada thread, é pedido o seu último bloco. Caso a resposta dos nós seja diferente de null, o bloco foi enviado é adicionado a um CopyOnWriteArrayList, e por fim, essa lista é retornada.
- **getCandidates():** Verifica todos os blocos da blockchain. Caso a fase do bloco seja a fase 1, pega nas transações de cada bloco e verifica quem foi o remetente dessa transação. Cada um desses remetentes terá recebido uma transação de 0 tokens do Master para registar na blockchain o candidato da eleição. Por fim, é retornado um CopyOnWriteArrayList com a lista de candidatos.
- **stateElector(String pubKey):** Recebe como parâmetro uma chave pública e verifica todas as transações de todos blocos da blockchain. Quando uma transação coincidir com a chave pública a verificar, atribui o valor 1 se for no destinatário da transação ou 2 se for no remetente da transação. Por fim retorna o valor obtido. Este valor serve para identificar na interface gráfica se o eleitor já votou ou não.
- **masterPublicKey():** Retorna a chave pública do Master que foi registada no bloco da fase 1. Para obter a chave pública do Master, são verificados os blocos da blockchain até achar o bloco

da fase 1, e é obtido o valor do Remetente dessa transação, que irá devolver a chave pública do Master. Por fim, essa chave é retornada.

- **masterPrivateKey():** Idêntico ao método masterPublicKey(), este método retorna a chave privada do Master que é registrada no bloco da fase 4. Para obter a chave pública do Master, são verificados os blocos da blockchain até achar o bloco da fase 4, e é obtido o valor do Remetente dessa transação, que irá devolver a chave pública do Master. Por fim, essa chave é retornada.
- **getBlockChain():** Retorna a blockchain.
- **getPhase():** Retorna a fase em que se encontra a eleição.
- **synchronizePhase(int phase):** Recebe como parâmetro um inteiro respeitante à fase a atribuir à eleição. Verifica se a eleição já se encontra nessa fase. Se sim, é feito o return para sair do método. Se não, atribui ao nó essa fase e sincroniza com os restantes nós da rede.
- **checkPhase(int phase):** Recebe como parâmetro um inteiro com a fase e verifica se é a mesma fase em que se encontra. Caso seja, retorna um booleano com essa informação.
- **getResults(String candidate, Private masterKey):** Recebe como parâmetros uma String com o nome do candidato e a chave privada do master. Verifica todos os blocos da blockchain, e se estiver na fase 4 (fase respeitante à divulgação da chave privada do master e disponibilização dos resultados da eleição) e o bloco pertencer à fase 3 (fase de votos por parte dos eleitores), itera todas as transações desse bloco, descripta as transações com a chave privada do master, e contabiliza em todas as transações as vezes que o candidato consta no destinatário da transação, que é a indicação de que o eleitor deu um token para esse candidato. Por fim, retorna a quantidade de tokens que o candidato tem.
- **setPhase(int phase):** Recebe como inteiro o valor da fase a atribuir à eleição. Se já estiver nessa fase, é feito o return para sair do método, se não, é atribuída essa fase e sincronizada com os restantes nós da rede.

7 Trabalho 4 – Segurança

7.1 Arquitetura da aplicação



Figura 10 - Diagrama de classes do package utils

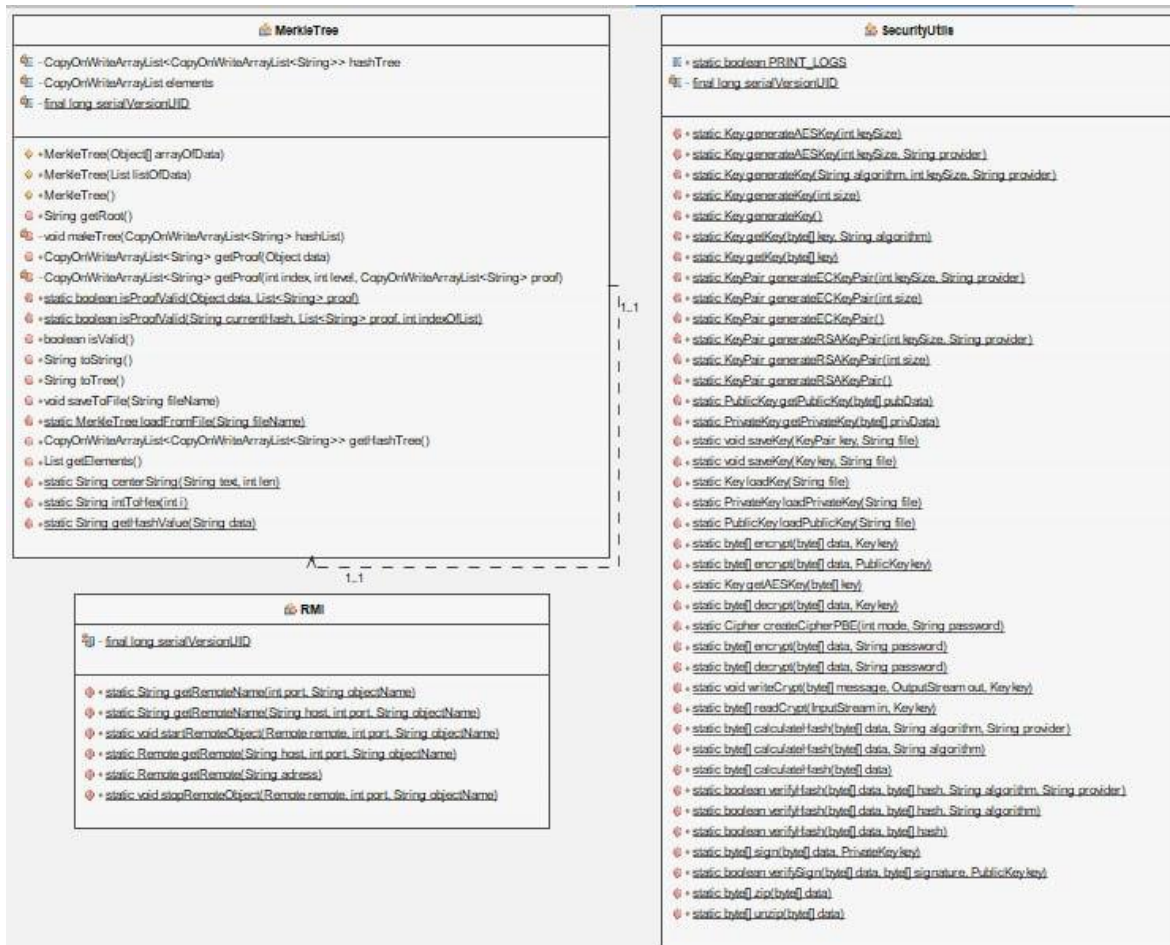


Figura 8 - Diagrama de classes do package utils

O Package *electionledger.utils* usa 2 classes, denominadas:

- **Credentials** – Classe que instância um utilizador com as suas chaves assimétricas e uma chave simétrica utilizada para fazer o registo e a autenticação no projeto. Diferencia um utilizador normal de um utilizador com nome “master” que servirá de utilizador autenticado para a comissão eleitoral que controla a eleição.
- **ImageUtils** – Esta classe contém os métodos estáticos para manipulação de imagens.
- **Hash** – Esta classe contém os métodos estáticos para obter uma hash de uma mensagem.
- **Converter** – Esta classe contém os métodos estáticos para conversão de dados entre objetos, byte arrays e hexadecimal.

- **Serializer** – Esta classe contém os métodos estáticos para conversão de dados entre objetos, byte arrays e hexadecimal.
- **GuiUtils** – Esta classe contém os métodos estáticos para a manipulação da interface.
- **RMI** – Esta classe contém os métodos estáticos para a utilização da biblioteca RMI do java.
- **SecurityUtils** – Esta classe contém os métodos estáticos para todos os métodos de encriptação e desencriptação com chaves assimétricas, simétricas e Strings.
- **MerkleTree** – Esta classe contém os métodos estáticos para a instanciação de uma árvore de merkle com obtenção da raiz.

Biblioteca electionledger. utils

Nesta biblioteca temos todos os métodos estáticos e classes auxiliares para a implementação do resto das bibliotecas, para conversão de dados, manipulação da interface e para manipular a encriptação de dados e uso de objetos remotos.

Para o trabalho de segurança vamos focar a classe credentials que vai implementar a segurança dos utilizadores que se autenticam e registam no projeto.

7.1.1 Classe Credentials.java

Esta classe permite o registo de utilizadores, um utilizador normal introduzindo o número de cartão de cidadão e uma password, ou um utilizador especial “master” que irá controlar a comissão eleitoral. A classe cria as chaves e os ficheiros para cada utilizador e de seguida pode autenticar o utilizador normal ou “master” a partir da password introduzida. A password irá desencriptar a chave simétrica que é desencriptada pela chave pública que depois desbloqueia a chave privada previamente desencriptada pela chave simétrica. Para o registo estes passos são feitos no sentido inverso, permitindo guardar a chave privada em ficheiro encriptada.

Uso de classes:

Usa a classe SecurityUtils para poder gerar as chaves assimétricas, a chave simétrica e para poder encriptar e desencriptar as strings e chaves geradas.

Atributos:**String:**

- **USER_PATH** – é o path do ficheiro que contém os utilizadores normais.
- **MASTER_PATH** – é o path do ficheiro que contém o utilizador master-
- **name** – nome do utilizador que irá ser master ou o número do cartão do cidadão.

PrivateKey:

- **privKey** – contém a chave assimétrica privada do utilizador.

PublicKey:

- **pubKey** – contém a chave assimétrica pública do utilizador

Key:

- **key** – contém a chave simétrica do utilizador

Métodos:

- **getName():** Devolve o nome do utilizador, que irá ser master, ou o número do cartão do cidadão do utilizador.
- **getPrivKey():** Devolve a chave assimétrica privada do utilizador.
- **getPubKey():** Devolve a chave assimétrica pública do utilizador.
- **getKey():** Devolve a chave simétrica do utilizador.
- **registarMaster(String password, String confirmpassword):** Recebe como parâmetro a password e de novo a mesma password para confirmação da introdução e cria um utilizador pré-definido como master, o método gera as chaves assimétricas e simétrica, encripta a chave privada com a password e de seguida gera a chave simétrica que é encriptada com a chave pública, estas chaves são de seguida guardadas no ficheiro master.
- **registar(String username, String password, String confirmpassword):** Recebe como parâmetro o número do cartão de cidadão, a password e de novo a mesma password para confirmação da introdução e cria um utilizado, o método gera as chaves assimétricas e simétrica, encripta a chave privada com a password e de seguida gera a chave simétrica que é encriptada com a chave pública, estas chaves são de seguida guardadas no ficheiro users.
- **autenticar(String username, String password):** Recebe como parâmetro o número do cartão de cidadão e a password, são acedidos os ficheiros de chave pública, simétrica e privada correspondentes ao username na pasta users e de seguida é desencriptada a chave privada com a password, que de seguida irá desencriptar a chave

simétrica, se isto acontecer com sucesso é devolvida uma instância do utilizador com as chaves simétricas e assimétricas escritas em aberto, descriptadas.

- **autenticarMaster(String password):** Recebe como parâmetro a password, são acedidos os ficheiros de chave pública, simétrica e privada correspondentes ao username master na pasta master e de seguida é descriptada a chave privada com a password, que de seguida irá descriptar a chave simétrica, se isto acontecer com sucesso é devolvida uma instância do utilizador master com as chaves simétricas e assimétricas escritas em aberto, descriptadas.

8 Interface Gráfica

8.1 Autenticação de Utilizadores

É a interface inicial para qualquer utilizador do sistema e permite o registo e autenticação para aceder à janela de eleitor ou a janela da comissão eleitoral.

8.1.1 Interfaces




Figura 11 Interface de Login

Esta interface recebe o username e password para autenticação do utilizador, se introduzir master com a password correta irá abrir a janela da comissão eleitoral, se

for um utilizador normal irá abrir a janela de eleitor. O login falha na introdução errada dos dados se não houver nenhum nó aberto na rede. Na parte de cima é atualizada a fase atual da eleição.



The image shows a web application interface for voter registration. The header is teal and contains the text 'Bem-Vindo Fase Atual: 1 - Registo de Eleitores' and a close button. Below the header, there are two main sections: 'Login' on the left and 'Registrar' on the right. The 'Registrar' section contains three input fields labeled 'Nº Cartão Cidadão', 'Escolha a sua Password', and 'Confirmar Password'. At the bottom of the 'Registrar' section is a button with a person icon and the text 'Confirmar Registo'.

Figura 12 Interface de registo.

Nesta interface é possível fazer o registo do eleitor ou master introduzindo o número de cartão de cidadão e a password com a confirmação da password.

Se se introduzir master no nome de utilizador e as passwords, cria-se um utilizador master.

Só é possível o acesso a registo a utilizadores normais se a fase mostrada acima for a fase de registo de eleitores.

8.1.2 Funcionalidades

8.2 Minerador da Rede

É a interface do miner e node da rede que utiliza o remoteObject programado no projeto, esta interface mina a os blocos da blockchain, sincroniza os nós da rede, permite conectar à rede e adicionar nós manualmente.

É apresentada toda a informação em tempo real relativa a conexões, transações e a blocos.

8.2.1 Interfaces

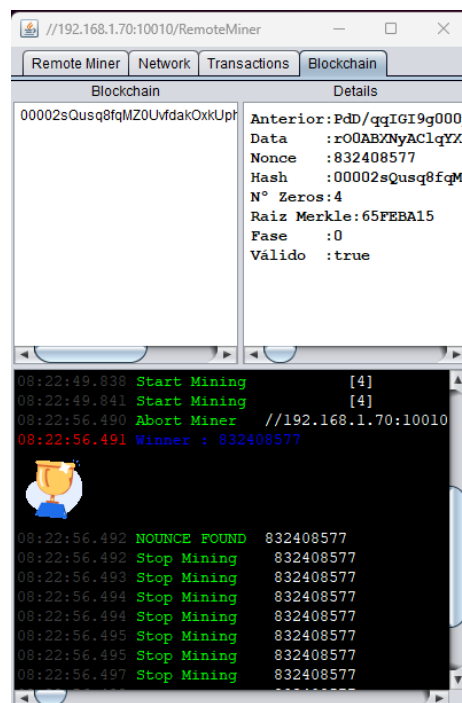


Figura 13 Interface Miner com bloco minado.

Esta interface mostra as transações que estão a ser minadas, a informação do bloco minado, e um log dos métodos de interface e soluções obtidas nos blocos.

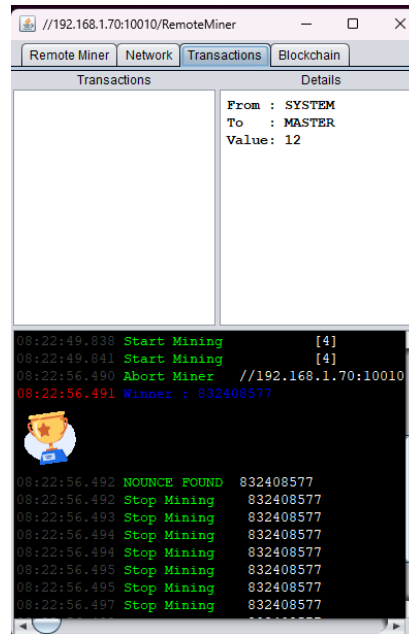


Figura 14 Interface de miner com informação de transações.

Esta interface mostra as transações que estão pendentes na rede.

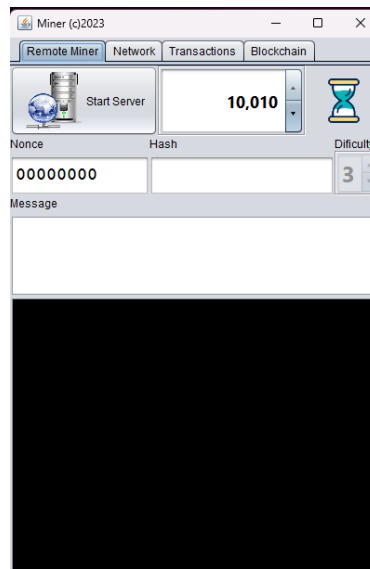


Figura 15 Interface de Miner para conexões.

Esta interface permite estabelecer a conexão aos nós da rede no domínio de Broadcast definido. É possível definir o port a ser usado e é apresentada a informação de conexões no log.

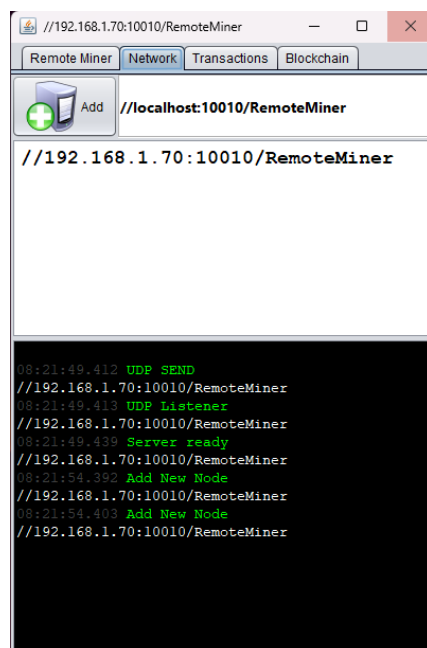


Figura 16 Interface de Miner com conexões na rede.

Esta interface mostra o estado da rede e os nós que estão conectados na rede, é permitido também a conexão manual a um endereço e port definido, porém esta implementação usa sockets e está a ser feita automaticamente para o domínio de Broadcast utilizado.

8.3 Utilizador “Master”

Esta é a interface do utilizador master que gere a eleição, a partir daqui é possível controlar a execução das fases da eleição, ver a informação e estados dos eleitores registados, verificar os resultados da eleição, ver o estado da blockchain, as ligações da interface à rede e a informação do utilizador master.

8.3.1 Interfaces



Figura 17 Interface da comissão eleitoral com o controlo das fases da eleição.

Esta interface mostra por ordem o controlo das fases da eleição;

- **Iniciar Eleição:** Recebe o parâmetro do número de utilizadores esperados para a eleição, ao clicar inicia a eleição com uma transação deste número de tokens para o master. Inicia a fase de registo de candidatos.
- **Confirmar Candidatos:** Aceita uma lista de candidatos manipulável, ao clicar Confirmar Candidatos é introduzido um bloco com a informação destes candidatos na blockchain e é iniciada a fase de registo de eleitores.
- **Validar Eleitores:** Ao clicar em Validar Eleitores são registados os eleitores aos quais são transferidos 1 token a cada e minado um bloco com estas transações.
- **Iniciar Votação:** Inicia a fase de votação que permite aos utilizadores introduzir o voto.
- **Terminar Votação:** Ao clicar em Terminar Votação é minado um bloco final que permite descriptar os votos da eleição com a chave privada da eleição.



Figura 18 Interface de comissão eleitoral que apresenta a informação dos eleitores.

Esta interface apresenta os eleitores registados e as suas chaves públicas, é atualizado em tempo real o registo dos eleitores e o estado dos eleitores como validados com “TOKEN TRANSFERIDO” escrito na entrada de cada um e “VOTO REGISTRADO” escrito na entrada de cada um.

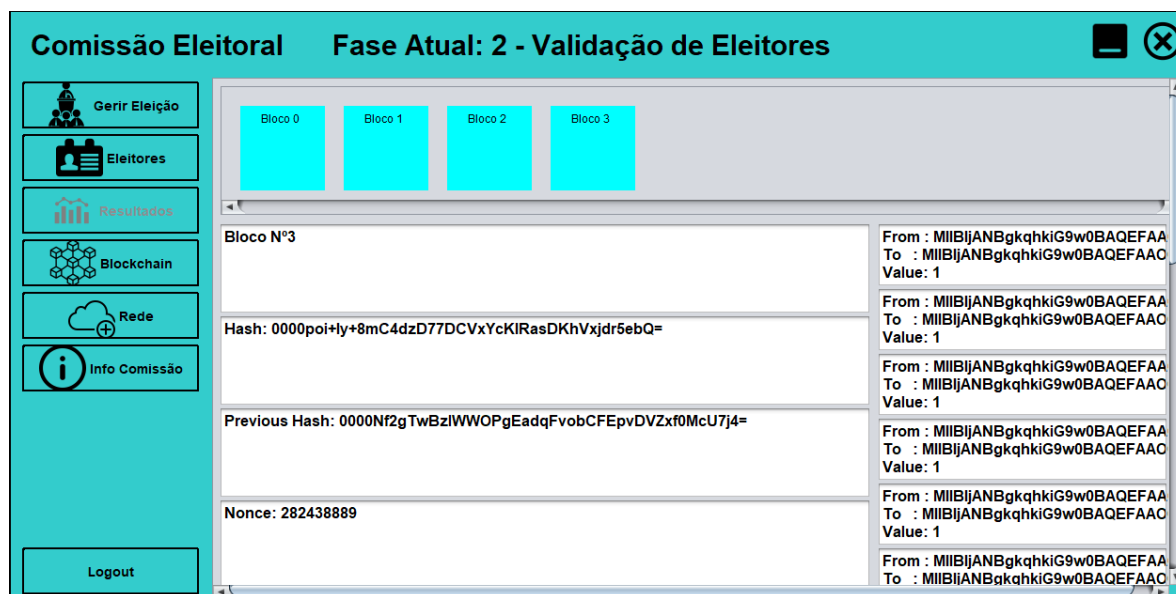


Figura 19 Interface da comissão eleitoral que permite explorar os blocos da blockchain.

Esta interface atualiza em tempo real os blocos que estão a ser introduzidos na blockchain, é permitido clicar no quadrado azul de cada bloco e apresentar a informação dos headers do bloco e as transações individuais em cada bloco.

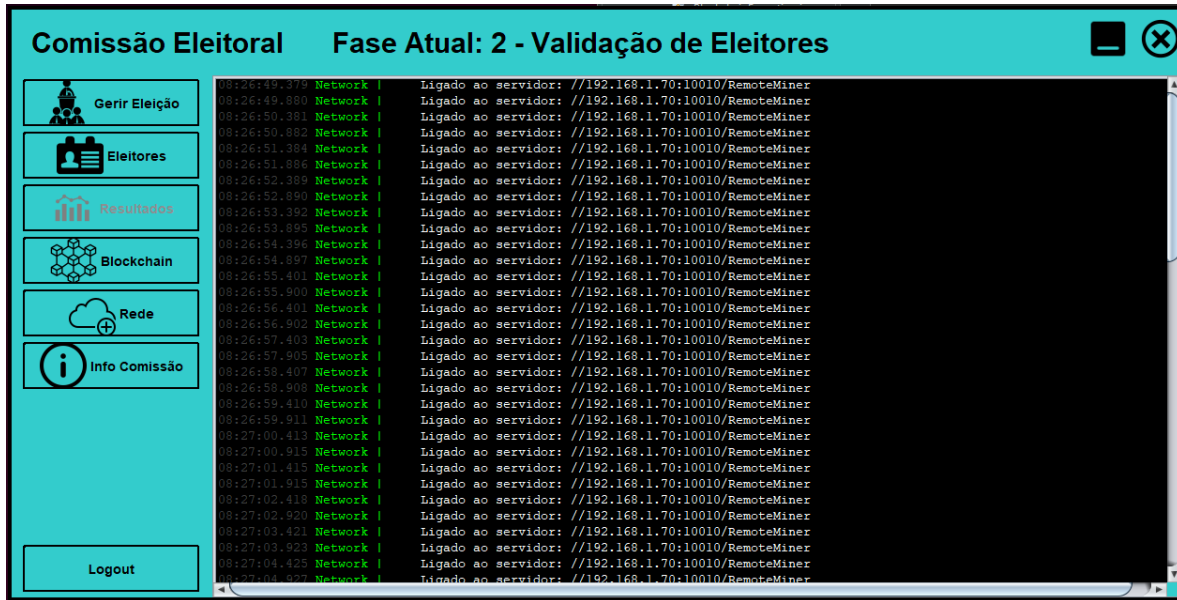


Figura 20 Interface da comissão eleitoral que apresenta logs da rede.

Esta interface mostra continuamente os nós que estão ligados à rede e permite verificar quando os nós são desligados.

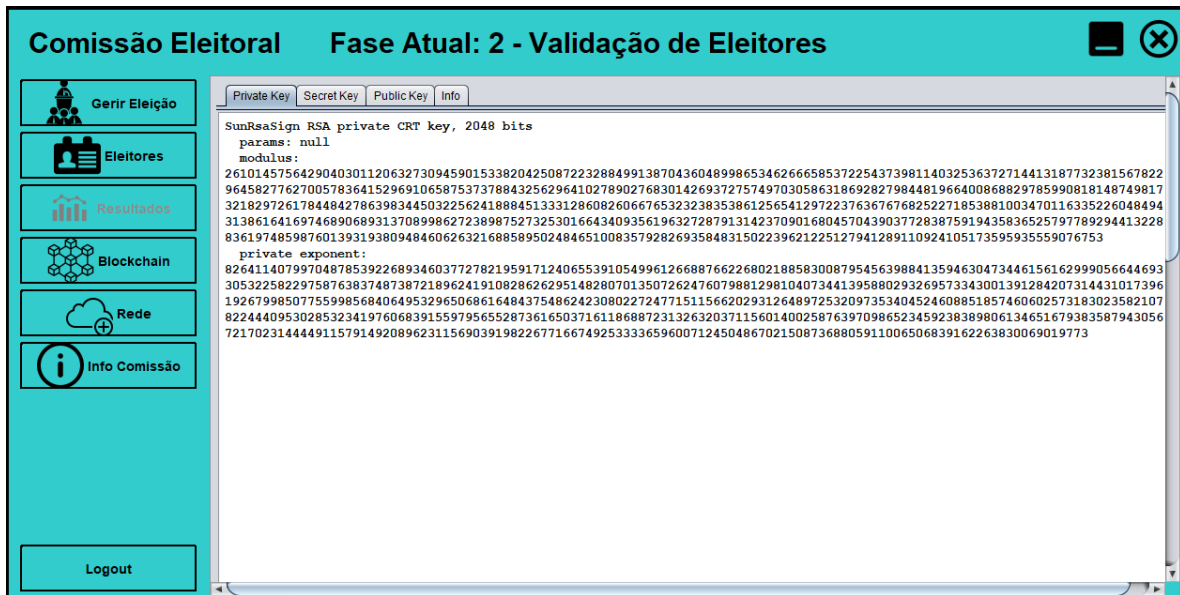


Figura 21 Interface da comissão eleitoral que permite ver informação do utilizador master.

Esta interface mostra a informação do utilizador autenticado incluindo a chave privada, a chave pública, a chave simétrica e o nome de utilizador.

8.4 Utilizador “Eleitor”

Esta interface é de um utilizador normal, ou eleitor, herda da interface da comissão eleitoral, os Resultados, Blockchain, Rede e Info.

É permitido o voto, a confirmação do voto e a apreentação de resultados ao eleitor.

8.4.1 Interfaces

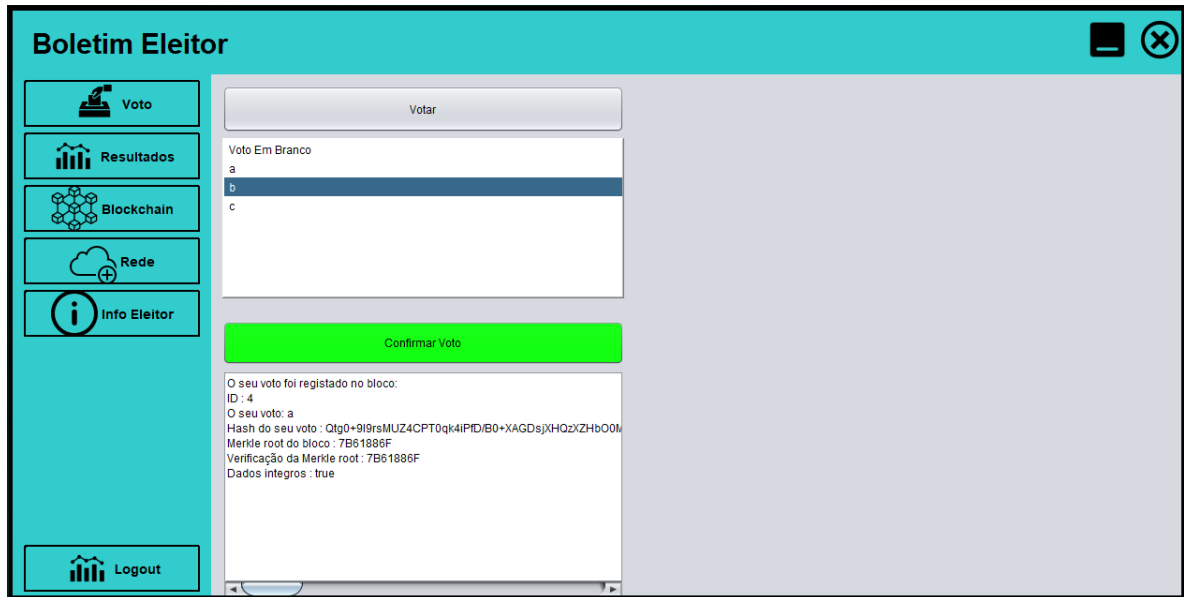


Figura 22 Interface de eleitor com janela de voto e confirmação.

Esta interface permite ao eleitor votar escolhendo um candidato, a funcionalidade só está disponível na fase de votação e não permite ao utilizador votar duas vezes. É possível também após a introdução e mineração do voto confirmar o voto, disponibilizando a informação do voto, o id do bloco, a validação da merkle tree do bloco e a validação da blockchain.



Figura 23 interface de eleitor com resultados da eleição

Esta interface mostra os resultados da eleição e só está disponível após a comissão eleitoral terminar a votação, é gerado um texto que conta os votos de cada candidato e um gráfico representativo dos votos da eleição.

9 Limitações e Desenvolvimentos Futuros

- A aplicação tem inconsistências a nível de apresentação de mensagens de erro e têm a particularidade de bloquear a interface do eleitor ou comissão quando é retirado um nó da rede em execução. Também não é permitido mudar a dificuldade dos blocos da eleição.
 - Testes mais extensos sobre as várias formas de quebrar o funcionamento normal da aplicação.

Por fazer:

- Guardar a blockchain em ficheiro e permitir aos miners fazerem backup quando são iniciados para que qualquer nó possa reiniciar a eleição se todos forem abaixo.
 - Existem os métodos para guardar a blockchain e sincronizar a blockchain, era necessário fazer com que cada minerador verifica-se a existência deste ficheiro localmente e tenta-se popular a rede e verificar se é o que tem o timestamp mais recente.
- Interface mais apresentável e intuitiva ou “user friendly”.
 - Como foi mais focado o conceito ambicioso de desenvolver as fases da eleição não focamos tanto a apresentação do projeto.
- Registo do master ficou pré programado para facilitação da defesa e apresentação do conceito do funcionamento.
 - Existem os métodos de registo do master porém quisemos pré programar esse utilizador para facilitar a defesa, porque isso também abre o conceito de cada master ter a sua eleição, e isso implicava que qualquer utilizador podia ser a comissão eleitoral no mesmo domínio de Broadcast e caberia aos utilizadores diferenciar qual a eleição correta a partir da chave pública do utilizador master da eleição correta e verificação de uma assinatura num bloco inicial.

Cada aplicação remete a uma eleição específica, ou então cabe a informação à parte da aplicação de cruzar os dados da chave pública da comissão eleitoral com o órgão confiável no mundo real.

- **Novas funcionalidades:**

- Para este projeto implementámos o conceito de votação por tokens tal e qual como transações e eleição por fases, o que ofereceu um desafio maior do que foi proposto, esta solução foi pesquisada na fase de pesquisa do estado da arte, entre outras soluções. O artigo científico que visa esta proposta é o seguinte:

Referência:

Blockchain-Enabled E-voting By: Nir Kshetri and Jeffrey Voas
Kshetri, Nir and Voas, J. (2018). " Blockchain-Enabled E-voting ", IEEE
Software 35(4), 95-99 Made available courtesy of IEEE:
<https://doi.org/10.1109/MS.2018.2801546>

A partir daqui vimos que isto já foi tentado e pareceu a melhor solução porque permite criar logo uma base que pode adicionar muito mais complexidade à segurança e privacidade.

10 Conclusão

Com este projeto conseguimos concretizar toda a matéria leccionada ao longo deste ano letivo e foi elaborada toda a infraestrutura que permite uma aplicação robusta dos conceitos de e-voting com segurança e privacidade, dado que quisemos adicionar complexidade ao projeto adicionando fases de registo e fases de validação de eleitores penso que foi um passo que nos deu mais compreensão da flexibilidade da blockchain que apesar de ser um conceito de descentralização, pode oferecer um controlo mais versátil na implementação do protocolo dos nós da rede.

Quisemos assim um protocolo que não só fizesse a mineração de blocos de forma sistemática, mas que houvesse uma lógica a ser dinamicamente implementada na blockchain o que já entra no campo de um protocolo que implementa um smart contract na base da lógica do protocolo em vez de executar as instruções numa linguagem de smart contract.

Em grupo fizemos um trabalho que encontrou dificuldades na parte de multithreading de forma distribuída e sincronização pelos nós da rede, mas estes blocos principais foram alcançados. Com esta ambição de funcionalidade, a parte visual e de experiência de utilizador ficaram mais para trás, porém na área desta cadeira achamos que permitir que os conceitos dados no ano letivo funcionassem todos em conjunto era um objeto mais claro e prioritário.

11 Referências

11.1 Documentos

- [D1] Nakamoto, Satoshi. "*Bitcoin: A peer-to-peer electronic cash system.*" *Decentralized Business Review* (2008)
- [D2] Ruhi Taş e Ömer Özgür Tanrıöver, "A Systematic Review of Challenges and Opportunities of Blockchain for E-Voting", Ankara University, 24, (2020)
- [D3] Nir Kshetri and Jeffrey Voas Kshetri, Nir and Voas, J., "Blockchain-Enabled E-voting", IEEE, 99, (2018)

11.2 Referência web

- [REF] Título, endereço, autores, data de consulta
- [W1] O que é e como funciona o blockchain: além das criptomoedas, URL: <https://distrito.me/blog/blockchain-o-que-e-como-funciona/>

Este trabalho foi efectuado e assinado digitalmente por todos os alunos do grupo, seguem as assinaturas de cada elemento do grupo: