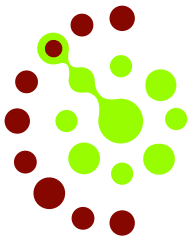

IPBeja

Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Blockchain E-Voting

Sistema de votação electrónica baseado em Hyperledger Fabric

António de Jesus Urbano Baião

Beja, 1 de Fevereiro de 2022

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Blockchain E-Voting

Sistema de votação electrónica baseado em Hyperledger Fabric

António de Jesus Urbano Baião

Orientado por :

Doutor José Jasnau Caeiro, IPBeja

Dissertação, realizado no Mestrado de Engenharia de Segurança Informática,
apresentado na
Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja

2022

Resumo

Blockchain E-Voting

Sistema de votação electrónica baseado em Hyperledger Fabric

Os sistemas de votação tradicionais têm sofrido problemas de confiabilidade, transparência e demora na obtenção dos resultados. Nos últimos vinte anos têm sido feitas várias tentativas de implementação de sistemas de votação electrónica. Para colmatar as falhas existentes nos sistemas tradicionais. No processo a diminuição das taxas de abstenção.

Estes sistemas padecem de algumas fragilidades devido fundamentalmente a serem sistemas centralizados, e controlados por uma só entidade. O surgimento do Blockchain tem vindo a trazer novidades no campo da votação electrónica. Este é um sistema que promete melhorias em relação aos sistemas de votação electrónica tradicional. Para desenvolver uma alternativa utilizou-se nesta dissertação a Framework Hyperledger Fabric. Esta permite criar redes distribuídas de Blockchain do tipo «Permissioned». Torna-se assim o processo eleitoral mais seguro, escalável, transparente, resiliente e fiável. Nesta dissertação apresenta-se, também uma aplicação para dispositivos electrónicos. Esta permite aos votantes interagir com a rede de Blockchain através de uma API de ligação. Pode desta forma o eleitor votar em qualquer lugar com comodidade e segurança.

Palavras-chave: *blockchain, votar, eleição, rede, Hyperledger Fabric, votante, votação electrónica.*

Abstract

Blockchain E-Voting

Sistema de votação electrónica baseado em Hyperledger Fabric

The traditional voting systems have been suffering since the beginning of reliability, transparency and delays in getting the results problems. In the last twenty years have been done a lot of tries to implement electronic voting systems. To fill the gaps of traditional systems. Trying during the process to decrease the abstention rate. These systems suffer from some weaknesses due to the fact they are centralized systems and be controlled for just one entity. The appearance of Blockchain has been bringing news in the field of electronic voting. This is a system that promises improvements regarding the traditional electronic voting systems. To develop a good alternative was used in this dissertation the Framework Hyperledger Fabric. This one, alow to create Blockchain distributed networks of «Permissioned» type. Becoming the election process safer, scalable, transparent, resilient and reliable. Along with an application for electronic devices. This one, allow the voters to interact with the Blockchain network through a communication API. The voters can this way vote in any place, with all the commodity and safety.

Keywords: *blockchain, vote, election, network, Hyperledger Fabric, voter, electronic voting.*

Agradecimentos

Agradeço à minha mãe e pai pelo apoio na minha constante busca pelo conhecimento. Agradeço ao professor Doutor José Caeiro por me ter guiado a bom porto no desenvolvimento desta dissertação e em todo o meu percurso académico. Ao meu colega e amigo Carlos Palma pela constante disponibilidade para debatermos e ganharmos conhecimento em conjunto. Agradeço também a todos os amigos e familiares que de alguma forma foram importantes para a realização deste mestrado.

Índice

Resumo i

Abstract iii

Agradecimentos v

Índice vii

Índice de Figuras xi

Índice de Tabelas xiii

1 Introdução 1

2 Estado da Arte 3

2.1 Introdução . . . 3
2.2 O que é o Blockchain? . . . 4
2.3 Onde e quando surgiu o Blockchain? . . . 4
2.4 Princípios de funcionamento do Blockchain. . . . 4
2.5 Os tipos de Blockchain. . . . 5
2.6 As Frameworks de Blockchain. . . . 6
2.7 História dos sistemas de votação eletrónica . . . 6
2.7.1 Os sistemas de votação eletrónica a usar o Blockchain. . . . 7
2.8 Conclusão . . . 9

3 Arquitetura do Sistema 11

3.1 Introdução . . . 11
3.2 O que é o Hyperledger Fabric . . . 11
3.2.1 Identity (Identidade) . . . 13
3.2.2 Membership Service Provider (MSP) . . . 13
3.2.3 Certificate Authorities (Autoridade Certificadora) . . . 13
3.2.4 Orderers . . . 14
3.2.5 Organizações . . . 14

3.2.6 Policies (Políticas) . . . 15
3.2.7 Legder (Livro Razão) . . . 15
3.2.8 Chaincode . . . 16
3.2.9 Peer . . . 16
3.2.10 Channel (Canal) . . . 17
3.2.11 Comunicação entre nós da rede . . . 18
3.2.12 Como se processa uma transação? . . . 18
3.3 Processo eleitoral . . . 20
3.3.1 Estrutura Aplicacional . . . 20
3.3.2 Divisão organizativa . . . 21
3.3.3 Processo de votação . . . 22
3.4 Conclusão . . . 26

4 Realização Experimental **29**
4.1 Introdução . . . 29
4.2 A rede de Blockchain . . . 29
4.3 O Minifabric . . . 30
4.3.1 Como utilizar . . . 31
4.3.2 Criação da rede . . . 32
4.3.3 Obter imagem Hyperledger Fabric . . . 33
4.3.4 Geração de certificados (certgen) . . . 33
4.3.5 Colocar em contentores Docker os ativos da rede (netup) . . . 35
4.3.6 Criação do canal (channelcreate) . . . 36
4.3.7 Juntar Peers ao canal (channeljoin) . . . 36
4.3.8 Define os Peers âncora. . . . 37
4.3.9 Criação de perfis de ligação (profilegen) . . . 37
4.3.10 Criação do Chaincode . . . 38
4.3.11 Instalar o Chaincode (ccinstall) . . . 40
4.3.12 Aprovação do Chaincode (ccaprove) . . . 41
4.3.13 Enviar a definição do Chaincode ao canal (cccommit) . . . 42
4.3.14 Instanciação do Chaincode (ccinstantiate) . . . 42
4.3.15 Serviço de Descoberta (discover) . . . 42
4.4 Criação de uma API usando o SDK . . . 43
4.4.1 Ligação à rede Blockchain . . . 43
4.4.2 Estrutura da API . . . 44
4.4.3 Função - enrollAdmin - API Administrativa . . . 44
4.4.4 Função - registerVoter - API Administrativa . . . 46
4.4.5 Função - createElection - API Administrativa . . . 48
4.4.6 Função - openElection - API Administrativa . . . 49
4.4.7 Função - closeElection - API Administrativa . . . 49

4.4.8 Função - login - API Cliente . . . 49
4.4.9 Função - isLoggedIn - API Cliente . . . 50
4.4.10 Função - getElectionResults - API Cliente . . . 50
4.4.11 Função - castBallot - API Cliente . . . 50
4.4.12 Função - getElections - API Cliente . . . 50
4.4.13 Função - getBallot - API Cliente . . . 50
4.5 Criação da aplicação cliente . . . 51
4.5.1 Página de login . . . 51
4.5.2 Página de listagem das eleições . . . 51
4.5.3 Página de Votação . . . 52
4.5.4 Página de visualização dos resultados . . . 52
4.6 Conclusão . . . 53

5 Conclusão **55**

Bibliografia **57**

Apêndices **59**

I Apendice - Hyperledger Fabric Scripts **61**

II Apendice - SDK APIs **93**

III Apendice - Chaincode **111**

Índice de Figuras

3.1 Esquema de uma rede Hyperledger Fabric 12
3.2 Identidades aceites 13
3.3 Emissão de certificados 14
3.4 Ledger, World State e Blockchain 16
3.5 Fluxograma de uma transação 18
3.6 Estrutura Organizativa da rede de Blockchain. 22
3.7 Estrutura Organizativa com representação dos Peers, Ledger e Chaincode. . . . 22
3.8 Processo de autenticação. 23
3.9 Processo de listagem das eleições. 24
3.10 Processo de submissão de voto. 25

4.1 Processo de submissão de voto 30
4.2 APIs, as suas funções e acesso à rede 45
4.3 Página de Login 51
4.4 Página de listagem das eleições 52
4.5 Página de votação 52
4.6 Página de listagem de eleições com possibilidade de ver resultados 53
4.7 Página de resultados de uma eleição 53

Índice de Tabelas

2.1 Comparativo de soluções existentes 7

Índice de Listagens

4.1 Ficheiro de criação da rede . . . 32
4.2 Função de criação de eleição do chaincode . . . 39
4.3 Carregar o ficheiro configurações da ligação . . . 43
4.4 Carrega a carteira e instância a Gateway. . . . 43
4.5 Tentativa de obtenção do utilizador da carteira . . . 44
4.6 Ligação ao canal e ao chaincode . . . 44
4.7 enrollAdmin - Criação do serviço CA para uso do recursos do CA . . . 45
4.8 enrollAdmin -Inscrição do administrador e gravação da entidade resultante na carteira . . . 46
4.9 registerVoter - Pedido de registo . . . 46
4.10 registerVoter - Registo do utilizador . . . 47
4.11 registerVoter - Pedido de registo . . . 48
I.1 Configuração para geração de material criptográfico . . . 61
I.2 Script de geração dos ficheiros usados para adesão das organizações . . . 62
I.3 Ficheiro usado para adesão da organização orq . . . 63
I.4 Script de geração para criação do canal . . . 67
I.5 Ficheiro de configuração obtido do Hyperledger Fabric . . . 67
I.6 Script de adição dos peers da organização orq ao canal . . . 86
I.7 Script de definição do peer ancora da organização. . . . 87
I.8 Script de instalção do chain code nos peers da organização ńorqż . . . 88
I.9 Script de aprovação do chaincode . . . 89
I.10 Script de envio da definição do chaincode ao canal . . . 91
II.1 API de suporte à aplicação cliente . . . 93
II.2 API de administração . . . 97
II.3 Mecanismo de licação à rede de blockchain. . . . 102
III.1 Chaincode . . . 111
III.2 Ballot . . . 128
III.3 Election . . . 129
III.4 User . . . 130
III.5 VotableItem . . . 131
III.6 Voter . . . 132

III.7 VoterEligible 134

Capítulo 1

Introdução

Esta dissertação aborda a criação de um sistema de votação eletrónica assente no Blockchain. Nos últimos vinte anos muito se têm questionado os processos eleitorais um pouco por todo o mundo. Muitos são acusados de falta de transparência, de falta de sigilo, fraude *etc*. O que muitas vezes faz com que as taxas de abstenção aumentem consideravelmente pondo em causa a confiança das pessoas no estado de direito. Como forma de mitigar este problema muitos estados começaram a investir em soluções de votação electrónica. Um dos primeiros países a adotar este sistema foi a Estónia e este sistema ainda é usado nos dias de hoje.

Apesar do sucesso de algumas soluções implementadas, estas ou são centralizadas ou «standalone» (o que implica que os votos sejam descarregados manualmente para um dispositivo de armazenamento). Este facto faz com que estas soluções ganhem desconfiança por parte da população. Um exemplo são as recentes eleições nos Estados Unidos da América. País em que é usado uma mistura de votação eletrónica, com votação em papel, com votos por correspondência e por e-mail. Há suspeitas que Donald Trump tenha mandado gerar votos fictícios em alguns dos estados americanos para com isso vencer as eleições. Para evitar estas situações e para que os sistemas de votação electrónica ganhem mais confiança pelos cidadãos têm sido criados sistemas com base no Blockchain.

Um dos primeiros usos do Blockchain foi na construção da primeira criptomoeda do mundo, o Bitcoin. Desde então outros usos têm-lhe sido dados, entre os quais os sistemas de votação electrónica. O Blockchain oferece descentralização, fiabilidade, transparência e segurança dos dados armazenados. São todas estas as características que faltavam para tornar os sistemas de votação electrónica existentes mais confiáveis por parte da população.

Ao longo dos anos Frameworks que permitem agilizar as implementações de redes de Blockchain começaram a surgir. Uma destas Frameworks é o Hyperledger Fabric. É nela que assenta a proposta de sistema de votação eletrónica que é tema da dissertação.

Esta proposta tem como objetivo o desenvolvimento de um sistema de votação eletrónica com cobertura nacional. Simultaneamente capaz de garantir a segurança, resiliência e transparência no decorrer de uma eleição. Este sistema é projetado baseando-se na es-

trutura administrativa de Portugal. A sua unidade mais pequena é a freguesia que é usada como ponto de entrada da rede. O sistema é composto por uma rede de Blockchain, uma API e uma aplicação cliente. Esta última serve de porta de entrada para os utilizadores que queiram interagir com a rede. As regras de funcionamento da eleição são definidas num «Chaincode» que está instalado nos «Peers» de cada freguesia. Este faz a ponte entre a rede de Blockchain e os utilizadores través de uma API que é acedida por uma aplicação cliente e fará a comunicação com a rede de Blockchain. É na aplicação cliente que os votantes podem consultar os resultados da eleições e exercer o seu direito de voto. Sempre com a garantia que o seu voto será imutável e anónimo durante todo o processo de votação.

Esta dissertação é composta por cinco capítulos. O capitulo 1 «Introdução». O capitulo 2, «Estado da Arte» aborda a tecnologia Blockchain e os seus usos mais comuns. Apresenta os sistemas de votação eletrónica e as propostas que usam um sistema de Blockchain. O capítulo 3, «Arquitectura do Sistema», detalha a Framework Hyperledger Fabric e como funciona. Neste capitulo é apresentada a arquitetura da rede que vai suportar o sistema de votação eletrónica. No capítulo 4, «Realização Experimental», são tidos em conta todos os esquemas arquiteturais exibidos no capítulo anterior para se criar a rede de Blockchain. Nesse processo são detalhados todos os passos realizados. É construída a API que dá suporte à aplicação cliente, fazendo a ligação à rede. Por fim o capitulo 5, «Conclusão», apresenta uma breve análise de todo o processo realizado. Enfoco nos trabalhos a desenvolver no futuro.

Capítulo 2

Estado da Arte

2.1 Introdução

O propósito principal da votação eletrónica é baixar a taxa de abstenção, permitindo o voto em qualquer local com o recurso a dispositivos eletrónicos (telemóvel, computador etc.). É, também, objetivo garantir a fiabilidade do ato eleitoral e evitar fraude na votação, assegurando transparência dos resultados.Em simultâneo garantindo resultados em tempo real.

Desde o aparecimento do Blockchain que novos sistemas de votação eletrónica têm sido propostos. Estes sistemas são desenhados para os mais diversos tipos de eleições. Sejam elas eleições escolares, empresariais ou nacionais. A sua maioria usa Frameworks baseadas na tecnologia Blockhain. Estas Frameworks agilizam o processo de implementação.

Foram analisados alguns artigos sobre diversos sistemas de votação eletrónica. Esta análise mostrou que existe uma vasta gama de propostas. Cada uma com uma abordagem diferente tendo sempre por base o padrão Blockchain.

Este capítulo começa por estudar o que é o Blockchain, na secção 2.2 «O que é o Blockchain?». Conta um pouco da sua história na secção 2.3 «Onde e quando surgiu o Blockchain?», e do seu funcionamento na secção 2.4 «Princípios de funcionamento do Blockchain». Na secção 2.5 «Os tipos de Blockchain», são apresentados os três principais tipos de Blockchain. De seguida são mencionadas algumas das Frameworks de Blockchain mais conhecidas na secção 2.6 «As Frameworks de Blockchain». É comentada a história dos primeiros sistemas de votação eletrónica na secção 2.7 «História dos sistemas de votação eletrónica». No final é feita uma analise cronológica de alguns dos sistemas implementados. Esta análise teve como base a análise de alguns artigos científicos sobre o tema. Uso dados como: a Framework usada e o número de transações que podem fazer. Por ultimo na secção 2.8 «Conclusão», é apresentada uma breve conclusão sobre a análise aqui realizada.

2.2 O que é o Blockchain?

O Blockchain[1] é um mecanismo capaz de transmitir informação de forma segura e inviolável. Esse mecanismo é composto por uma cadeia de blocos encadeados. Estes certificam-se uns aos outros de forma a garantir que todos os dados neles contidos são válidos ao longo de toda a cadeia, usando códigos «hash» de cada bloco associados ao bloco anterior.

2.3 Onde e quando surgiu o Blockchain?

Há alguma incerteza sobre a forma como surgiu o Blockchain e muitas hipóteses foram escritas. Muitos nomes foram dados para a autoria. A hipótese mais recorrente sobre a data é que o Blockchain foi anunciado aquando da queda do banco Lehman Brothers nos Estados Unidos que acabou por ser o início do colapso da economia mundial em 2008. Consta que nessa altura surgiu nas redes sociais uma mensagem de alguém que se intitulava de Satoshi Nakamoto, dizia ter inventado um sistema de transferência de informação totalmente descentralizado, fiável e inviolável. Com base nesse sistema deu também a conhecer uma nova moeda em formato digital. O Bitcoin é, alegadamente, originado pelo senhor Satoshi Nakamoto. Há quem diga que a Bitcoin está intrinsecamente associada à criação do Blockchain.

2.4 Princípios de funcionamento do Blockchain.

O Blockchain é constituído por uma cadeia de blocos encadeados. Este tem como principal elemento o **bloco**. O bloco é constituído por um conjunto de dados que se quer transmitir. Incorpora um código «hash» que garante a impressão digital desse bloco tornando o único. Está também presente a «hash» (impressão digital) do bloco anterior. Por norma o primeiro **bloco** da cadeia não vai ter «hash» referente ao bloco anterior e é chamado de bloco **génesis**. Todos os blocos criados daí para a frente têm sempre como base a impressão digital do bloco anterior, tornando a cadeia inviolável. Em resumo, o bloco contém a sua impressão digital, criada com base na «hash» anterior e nos dados contidos. Se algum dos blocos da cadeia for adulterado toda a cadeia daí em diante fica invalidada.

Um sistema de Blockchain é constituído por vários nós (máquinas) que contêm uma cópia do Blockchain (Ledger). Qualquer nó do sistema pode criar novos blocos. Contudo deverão existir mecanismos para que não seja fácil criar novos blocos, para evitar que qualquer novo bloco seja adicionado na cadeia sem consentimento de todos na rede. Para isso existem critérios de segurança a serem cumpridos. Esses critérios podem ser o consenso da rede em validar o bloco e o cumprimento de regras de criação. Na rede de Bitcoin, por exemplo, obriga-se que exista uma certa quantidade de bits de valor 0 no início da «hash»

[1] What is a Blockchain?: https://hyperledger-fabric.readthedocs.io/en/release-2.4/blockchain.html

para que o bloco seja valido. Por forma a cumprir essas regras sem adulterar os dados a serem transmitidos é usando o campo suplementar chamado de «nonce». Este vai sendo incrementado até que a «hash» produzida cumpra os requisitos. Essa acção é designada de «Miner». Esta é também uma forma de garantir que não são adicionados novos blocos à rede de forma tão rápida e garantir que em caso de fraude não seja possível em tempo útil o recálculo das «hashs» de toda a cadeia.

Outro dos benefícios desta tecnologia é o uso de um «Ledger» distribuído. Há replicação de toda a cadeia por vários nós da rede tornando ainda mais difícil a adulteração dos dados. O «Ledger» é um registo que é público e partilhado. Usando um paradigma de P2P (Peer-to-Peer) o «Ledger» não precisa de uma unidade central para certificar que a cadeia de blocos de cada nó é válida. Essa certificação é obtida tal como num sistema democrático, com o consenso da maioria dos nós. Depois de validados pelos nós da rede, um novo bloco é aceite ou descartado consoante a deliberação de cada nó ao aplicar as regras.

2.5 Os tipos de Blockchain.

Existem três tipos de Blockchain[2], sendo o primeiro deles o Blockchain do tipo público que usa um «Ledger» público. Neste qualquer pessoa pode fazer parte da rede de Blockchain, ou seja, qualquer pessoa pode ser um novo nó da rede. Um dos exemplos disso é a criptomoeda Bitcoin, tal como são grande parte das criptomoedas.

Outro tipo de Blockchain, é em tudo semelhante ao público. Com o acrescento de uma camada de segurança que trata de identificar os utilizadores, bem como definir «roles» de utilizadores. Não é qualquer pessoa que pode entrar na rede. As pessoas com permissão para entrar na rede terão acções bem definidas com base no «role» em que se encontram. Esses «role» podem inclusivé controlar as partes do bloco que são visíveis e as que são sigilosas. Este tipo de Blockchain é designado por«Permissioned».

Por fim temos o Blockchain privado que consiste num determinado número de nós que são previamente escolhidos. Este é usado em redes internas. É usado por exemplo em instituições bancárias, onde é preciso ter um sistema de Blockchain para registo e segurança de todas as transacções envolvidas. Não é permitido que qualquer pessoa possa fazer parte da rede.

[2]What is a Blockchain?: https://hyperledger-fabric.readthedocs.io/en/release-2.4/blockchain.html

2.6 As Frameworks de Blockchain.

Existem várias Frameworks de Blockchain[3] que facilitam a criação de novas redes. Uma dessas Frameworks baseada no Blockchain público é o Ethereum[4]. O Ethereum é uma plataforma descentralizada capaz de executar contratos inteligentes e aplicações descentralizadas usando a tecnologia do Blockchain. Estes funcionam exatamente como programados sem quaisquer possibilidade de censura, fraude ou interferência de terceiros. Isto porque o contrato é imutável. Esta Framework foi lançada em 2015[5] e usa como base a linguagem de programação Solidity[6] que foi criada para escrever os contratos inteligentes.

O Hyperledger Fabric[7] é uma Framework modular que funciona como fundação para desenvolver produtos baseados em Blockchain. Soluções e aplicações que usem componentes *Plug and Play* destinados ao uso de empresas privadas. O Hyperledger[8] é um projecto colaborativo entre várias indústrias e foi iniciado em Dezembro de 2015 pela Linux Foundation. Tem como objetivo avançar a tecnologia do registo distribuído de Blockchain em múltiplos segmentos da indústria. Dentro da família Hyperledger existem outros projectos de Blockchain. Esta é uma Framework de Blockchain *Permissioned*.

2.7 História dos sistemas de votação eletrónica

A votação é essencial para qualquer democracia moderna. Actualmente os sistemas de votação eletrónica são usados nas eleições nacionais de cerca de 15,7%[9] dos países do mundo. Eleições recentes[10] testemunharam o aumento de votos electrónicos e o aumento da confiança que as pessoas têm pelo sistema de votação eletrónica. A Estónia foi a primeira nação do mundo a ter o seu sistema de votação totalmente eletrónico que está a funcionar desde 2005[11].

Tradicionalmente os sistemas de votação são criticados por serem demasiado lentos e ineficientes. Por forma a resolver isto houve um grande interesse em criar sistemas digitais que ajudassem na contagem e na obtenção dos votos. Um processo conhecido como votação eletrónica. As primeiras gerações de sistemas de votação eletrónica eram centralizados e não resolviam os problemas de confiabilidade. Estas dependiam de um sistema terceiro e centralizado para auxiliar a obtenção e cálculo dos votos. Além disso a primeira geração de

[3] Top Blockchain Plataforms: https://www.leewayhertz.com/blockchain-platforms-for-top-blockchain-companies/
[4] Ethereum: https://ethereum.org/en/
[5] History of Ethereum: https://ethereum.org/en/whitepaper/
[6] Solidity documentation: https://docs.soliditylang.org/en/latest/
[7] Hyperledger Fabric: https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf
[8] Hyperledger: https://www.hyperledger.org/
[9] E-voting by country: https://www.idea.int/data-tools/question-view/742
[10] Electronic Voting around the World: https://www.jstor.org/stable/pdf/resrep03645.9.pdf
[11] Electronic voting: What Europe can learn from Estonia: https://blogs.microsoft.com/eupolicy/2019/05/10/electronic-voting-estonia/

sistemas de votação eletrónica tinha falhas de rastreabilidade, verificabilidade, integridade e privacidade dos votantes que ainda hoje se verificam um pouco por todo o mundo[12].

2.7.1 Os sistemas de votação eletrónica a usar o Blockchain.

Desde a aparição do Blockchain que têm surgido novos objectivos para o seu uso. Nos últimos anos têm sido feitas tentativas de desenvolver sistemas de votação eletrónica baseadas no Blockchain. Em tabela2.1, estão apresentados alguns artigos que propõem novos sistemas de votação eletrónica [Abu+19].

Tabela 2.1: Comparativo de soluções existentes

Nome Artigo	**Ano**	**Nome da Aplicação**	**Framework**	**Escala**
Moscow Mayor official website	2017	Active Citizen	Ethereum	220 mil +
Blockchain-Enabled E-Voting [KV18]	2018	BEV (Blockchain enabled e-voting)	Própria	Pequena escala
Agora	2018	Agora	Própria	Larga escala
Votereum: An Ethereum-based E-voting system [Thu+19]	2019	Votereum	Ethereum	Media escala
Decentralized E-Voting Portal Using Blockchain [PJ19]	2019	N/A	Própria	Pequena escala (100 +)
VoteChain: A Blockchain Based E-Voting System [PBC19]	2019	VoteChain	Própria	Pequena escala (hipotéticamente larga escala)

2017 - Active Citizen

Em 2014 foi introduzido o sistema de votação eletrónica «Active Citizen» [13] comunitário na cidade de Moscovo. Este permite aos cidadãos ter voto no processo decisório da sua cidade. Estas decisões vão desde de manifestar a sua opinião na construção de uma nova ciclovia, até às cores dos assentos do novo estádio da cidade. Contudo desde cedo que o sistema esteve envolto em problemas de confiabilidade e falta de transparência, com a

[12] Voto eletrónico, eleições europeias, distrito e Évora: https://www.cnpd.pt/comunicacao-publica/noticias/voto-eletronico-cnpd-defende-rigoroso-escrutinio/

[13] Moscow introduces blockchain e-voting: https://www.smartcitiesworld.net/news/news/moscow-introduces-blockchain-e-voting-2365

população a criticar a forma e a veracidade dos votos recolhidos. Por forma a eliminar estes factores de desconfiança, o departamento dos sistemas de informação da cidade, decidiu em 2017 começar a tratar o processo de votação com recurso à tecnologia de Blockchain. Usa um Blockchain baseado na plataforma Ethereum. Desta forma os cidadãos conseguem ter acesso aos dados em tempo real, aumentando o grau de confiabilidade no sistema. Foi também dada a possibilidade aos cidadãos de fazerem parte da rede de Blockchain. Para tal ser possível, foi disponibilizado um manual de instruções que descreve como um cidadão pode usar o seu telefone para se tornar um nó da rede. Pode assim, fazer uma auditoria à votação. De forma estatística estima-se que desde o inicio do projecto 1.9 milhões de utilizadores e cerca de 2800 votações foram realizadas. Durante os picos de afluência foram processados 1000 votos por minuto.

2018 - Agora

O «Agora» [14] é um sistema de votação eletrónica baseado em Blockchain. Este usa um sistema criptograficamente seguro, distribuído e descentralizado. Os dados da eleição são armazenados num «Ledger» publico acessível a todos. Torna possível a eleição ser auditada por qualquer pessoa. Permite ainda aos votantes verificarem se o seu voto foi correctamente registado e se permanece inalterado. Promete que todos os dados relativos à eleição estão cifrados e anonimizados. Os votos podem ser submetidos a partir de qualquer dispositivo com acesso à Internet.

O mecanismo de consenso oferece alto rendimento e validação das transacções de forma eficiente. Este permite que até mesmo clientes com recursos limitados, como telemóveis, participem.

Uma comunidade global de operadores de nós organizados seguindo um modelo híbrido de permissão/sem permissão pode verificar os resultados da eleição pelo token VOTE.

1. **Nós de Consenso:** Trata-se dos nós de uma rede de Blockchain do tipo «Permissioned» e distribuída de organizações apartidarias. Estas fornecem consenso às transacções.

2. **Nós de audição dos cidadãos**: Uma rede global descentralizada de nós sem confiança que observa e verifica se o consenso é processado correctamente.

2019 - Votechain

A aplicação VoteChain, é uma prova de conceito para eleições em larga escala que foi realizada na Índia. No mesmo bloco existem vários votos. Usa o mesmo principio da

[14] Agora: https://impakter.com/agora/

mineração das Bitcoins. Delegando essa acção nas organizações que realizam a eleição ou em voluntários que queiram participar no processo. Os primeiros testes foram realizados numa eleição escolar, com cerca de 100 votantes [PBC19].

2019 - Votereum

Um prototipo de votação eletrónica, criado no Vietnam. Usa como base a Framework Ethereum. A aplicação foi implementada usando NodeJs para a criação de serviços web. Um sistema de filas RabbitMQ e uma interface Angular. É anunciado que é possível realizar mais de 45000 transacções num só dia (com base nas capacidades do Ethereum) [Thu+19].

Outros Sistemas

Um pouco por todo o mundo vão surgindo cada vez mais proposta de votação eletrónica. Estas prometem tornar as eleições fiáveis, seguras e transparentes. Esses sistemas lidam todos com um problema em comum. A quantidade de transacções que é possível realizar em tempo útil. Grande parte delas faz uso da Framework Ethereum [PJ19; KV18; Hja+18; Gao+19; BCD19].

2.8 Conclusão

São muitos os usos para o sistema de Blockchain nos dias de hoje. A sua capacidade de assegurar a imutabilidade dos dados torna o único no desenvolvimento de aplicações seguras, fiáveis e transparentes. Tem sido por isso cada vez mais frequente o aparecimento de novas soluções de votação eletrónica baseadas nesta tecnologia. Numa tentativa da população mundial ganhar confiança nos sistemas de votação eletrónica. Depois da analise feita aos diversos artigos conclui-se que existe uma grande diversidade de propostas. Nenhuma delas no entanto cem porcento perfeita. Abrindo espaço para novas abordagens e trilhagem de novos caminhos, no sentido de tornar os sistemas de votação eletrónica uma realidade das sociedades modernas.

Tendo isto em conta foi escolhida a Framework Hyperledger Fabric para criação do sistema de votação eletrónica aqui proposto. O sistema de consenso da rede é o que torna as redes geradas pelo Hyperledger Fabric diferenciadoras dos restantes sistemas de Blockchain distribuído. Durante a criação e submissão de transações para uma rede de Blockchain é necessário que todos os nós acordem entre si a validade da transação. Nos sistemas de Blockchain tradicionais, é usado processamento «Prof-of-work» (por exemplo a mineração que é realizada nos sistemas de criptomoedas) o que faz com que todo o processo seja moroso. Um sistema de votação eletrónica não pode aguardar largos minutos para que um voto seja aceite pelo sistema. Desta forma os sistemas que usam certificados digitais como forma de consenso são os mais indicados. Na rede do Hyperledger Fabric todos

os intervenientes, desde o cliente até ao nó da rede tem uma entidade composta por um certificado digital. Esta entidade é previamente conhecida pela rede. Esta característica faz com que as transações assinadas digitalmente sejam submetidas sem que haja «Prof-of-work» envolvido uma vez que todos os intervenientes da rede se conhecem entre si. Torna-se assim o processo eleitoral, mais rápido, menos dispendioso e sem a necessidade de elevados recursos de *hardware* como acontece com os sistemas tradicionais de Blockchain.

Capítulo 3

Arquitetura do Sistema

3.1 Introdução

Um dos principais pontos de desconfiança prende-se com a segurança dos dados. Tornar os dados seguros de forma a que estes não sofram adulterações durante o processo eleitoral, é fundamental para que a eleição seja considerada justa e os seus resultados confiáveis. Para responder a essa necessidade o sistema foi construído em uma plataforma de Blockchain.

Na implementação, foi utilizada a Framework Hyperledger Fabric. O sistema aqui descrito baseia-se nas eleições presidenciais portuguesas. A construção de uma rede de Blockchain que permita a realização da eleição é o ponto fundamental para aumentar a resiliência, confiabilidade e segurança dos dados. A rede é composta por organizações que representam as freguesias de Portugal.

Este capítulo tem como objetivo descrever como com recurso à Framework Hyperledger Fabric, foi construído o sistema de votação eletrónica. Este é robusto, escalável e confiável.

Na secção 3.2 «O que é o Hyperledger Fabric», é descrito o Hyperledger Fabric e o seu funcionamento geral. Explicam-se todos os seus componentes principais necessários para a execução deste sistema. Na secção 3.3 «Processo eleitoral», está descrito o sistema de votação eletrónica. Nomeadamente como é feita a divisão organizativa da eleição presidencial. Apresenta-se, também, o fluxo de dados gerados por todas as ações permitidas pela aplicação. Na secção 3.4 «Conclusão», as conclusões de como a estrutura e modo de funcionamento do Hyperledger Fabric encaixam na implementação do sistema de votação eletrónica.

3.2 O que é o Hyperledger Fabric

O Hyperledger Fabric[1] é uma tecnologia *open source* que permite criar redes de Blockchain do tipo «Permissioned» de forma distribuída para uso empresarial. Esta tecnologia é diferenciadora quando comparada com os mais populares sistemas de Blockchain distribuídos

[1]Hyperledger Fabric documentation: https://hyperledger-fabric.readthedocs.io/en/latest/

ou outras plataformas de Blockchain. Isto deve-se à forma com se realiza o consenso entre os nós da rede.

O Hyperledger foi criado pela Linux Foundation em 2015. O seu desenvolvimento é apoiado pela comunidade da qual fazem parte cerca de 35 organizações e perto de 200 programadores. O Hyperledger Fabric é um dos seus projetos de Blockchain.

Existem outros projectos[2] baseados na tecnologia Blockchain, como o Hyperledger Iroha usado em projetos de IoT ou de infraestruturas. Hyperledger Indy, Hyperledger Besu baseado em Ethereum, Hyperledger Burrow e Hyperledger Sawtooth. Todos eles com caracteristicas distintas mas tendo em comum o uso de um «Ledger» distribuído.

Esta tecnologia é altamente modular e configurável sendo facilmente adaptável a uma ampla gama de casos. A banca, seguros, sistemas de saúde, gestão de recursos humanos entre outros.

É uma plataforma desenhada para a construção de redes de Blockchain distribuídas. Está baseada numa arquitetura que lhe confia um elevado grau de confiabilidade, resiliência, flexibilidade e escalabilidade.

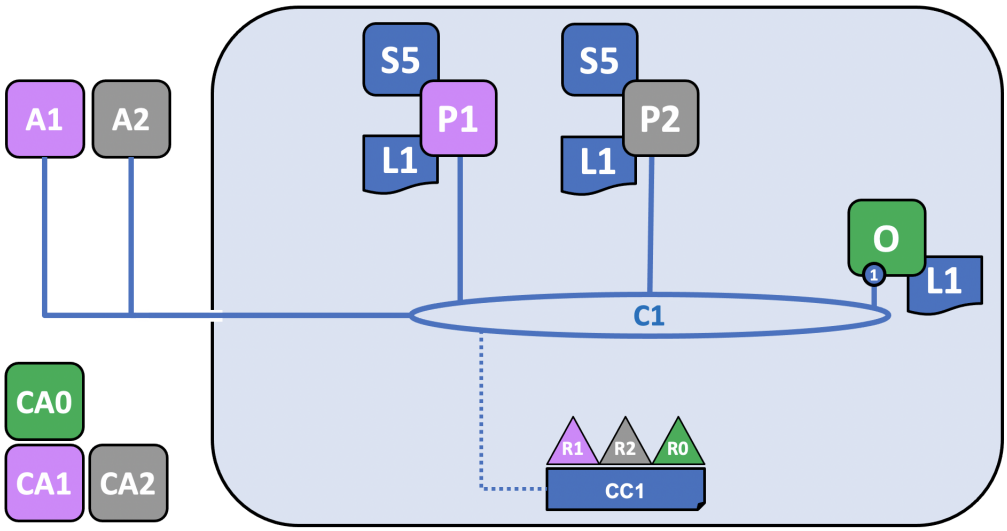

Figura 3.1: Esquema de uma rede Hyperledger Fabric

(Imagem retirada da documentação do Hyperledger Fabric https://hyperledger-fabric.readthedocs.io/en/release-2.2/network/network.html)

De um modo geral uma rede criada pelo Hyperledger Fabric é constituída por *Orderers*, *Peers*, *MSPs*, Organizações, Canais, *CAs*, *Ledgers* e Clientes. Na figura 3.1 está representado o esquema arquitetural de uma rede padrão.

[2]Distributed Ledgers Hyperledger projects: https://www.hyperledger.org/use/distributed-ledgers

3.2.1 Identity (Identidade)

A identidade compreende os diferentes atores de uma rede de *Blockchain*. Os «*Peers*» , os «*Orderers*» e os clientes das aplicações. Cada um destes atores tem uma identidade digital encapsulada num certificado X.509. As identidades são responsáveis por determinar as permissões sobre os recursos e o acesso à informação que os atores têm na rede. Identificar os atores perante outros atores da rede. A identidade só é valida se for emitida por um «*MSP*» .

Figura 3.2: Identidades aceites

(Imagem retirada da documentação do Hyperledger Fabric
https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html)

3.2.2 Membership Service Provider (MSP)

O «Membershsip Service Provider» (MSP) é o componente que define as regras pelas quais as identidades são validadas e autenticadas. Este também define quais as permissões que essas identidades têm no acesso à rede. Gere também os Ids dos utilizadores e autentica clientes que se querem juntar à rede. Isto inclui fornecer credenciais para os clientes proporem transações. O «MSP» faz uso de uma Autoridade Certificadora («Certificate Authority») conhecida como «CA» , que é capaz de verificar e revogar os certificados de um dado utilizador após a confirmação da sua identidade. O «MSP» usa por omissão o standard X.509 que utiliza o tradicional esquema de PKI (Infraestrutura de Chave Publica)[3]

3.2.3 Certificate Authorities (Autoridade Certificadora)

Os «CAs» (Certificate Authorities) são parte comum dos protocolos de segurança. As mais conhecidas são Symantec (originalmente Verisign), GeoTrust, DigiCert, GoDaddy, e Comodo entre outros.

[3]PKI: https://hyperledger-fabric.readthedocs.io/en/release-2.2/identity/identity.html

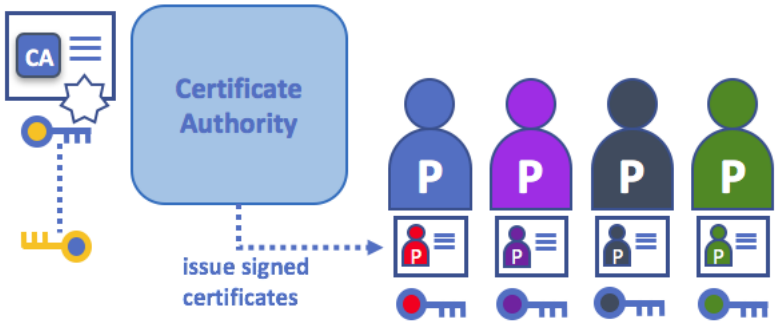

Figura 3.3: Emissão de certificados

(Imagem retirada da documentação do Hyperledger Fabric https://hyperledger-fabric.readthedocs.io/en/release-2.2/identity/identity.html)

O Hyperledger Fabric fornece um componente que permite criar «CAs» na rede de Blockchain. O componente conhecido como «Fabric CA» é um «CA Root» privado. Permite que a partir dele sejam criadas autoridades certificadoras intermédias. Garante assim a cadeia de confiabilidade sempre que um certificado for gerado. Fornece a capacidade de gerir as entidades do Hyperledger Fabric que tenham como padrão certificados do tipo X.509.

3.2.4 Orderers

Também conhecidos como «Ordering Service» (Serviços de Ordenação). Assinalados na figura 3.1 como «O». Trata-se de um conjunto de «nós» responsáveis por ordenar as transações num bloco e distribuir os blocos pelos «Peers» ligados. As transações são ordenadas na lógica de «First-come-first-serve». Os serviços de ordenação tem conhecimento da identidade de todos os membros da rede sendo capazes de forma segura de efetuar transações com estes.

3.2.5 Organizações

Uma organização é responsável por gerir grupos de entidades que podem interagir com a rede de Blockchain. As organizações estão representadas na figura 3.1 como «R». Estas entidades podem ser «CAs», «Orderers», «Peers» entre outros. Uma organização pode juntar-se a uma rede depois de dar a conhecer o seu «MSP» (Membership Service Provider). O «MSP» define como os outros membros da rede podem verificar que as assinaturas (tais como as usadas para identificar as transações) foram geradas por uma entidade valida e emitidas pela organização. Os direitos de acesso das identidades de um «MSP» são geridos por políticas que são acordadas quando a organização se junta à rede. Uma organi-

zação não tem limite de tamanho podendo ser constituída por «N» entidades. Os «Peers» são os pontos responsáveis por processar as transações recebidas pela organização.

3.2.6 Policies (Políticas)

Uma política é um conjunto de regras que define a estrutura de decisão e os objetivos alcançados. As políticas estão representadas na figura 3.1 como «CC» As políticas são descritas como «quem», «o quê», bem como o acesso e os direitos que alguém tem sobre um ativo da rede.

As políticas são usadas como mecanismo de gestão da infraestrutura. Estas representam como os membros chegam a acordo no que diz respeito a aceitar ou rejeitar alterações na rede, canal ou no «Chaincode». As políticas são acordadas por todos os membros do canal quando este é criado. Estas descrevem o critério de adição ou remoção de membros de um canal e alterações à forma como os blocos são formados. Especificam um número de organizações requeridas para realizar o «Endorsement» [4] num chaincode entre outras ações.

3.2.7 Legder (Livro Razão)

Um *Ledger* contém o estado atual de um determinado negócio como um diário de transações. O *Ledger* está representado na figura 3.1 como «L». O conceito mais similar com um *Ledger* é o registo do estado e o extrato de uma conta bancária. Em que o estado é indicado pelo saldo contabilístico. O extrato dá-nos os históricos de transações (créditos e débitos) que conduzem aquele saldo atual (estado atual). De certa forma o Hyperledger Fabric assenta no mesmo princípio quanto à forma como o seu *Ledger* é construído.

O «*Ledger*» é constituído por duas partes distintas. Um «*World State*» e um *Blockchain*.

O «***World State***», o estado atual de um elemento, é normalmente armazenado numa base de dados. Tornando possível aceder ao estado atual de um elemento armazenado no *Blockchain* sem ter de o percorrer por completo para determinar o estado atual. Os elementos armazenados no «*Ledger*» são normalmente representados por um conjunto chave-valor. O «*World State*» pode ser frequentemente alterado porque os elementos podem ser criados, atualizados ou apagados.

O ***Blockchain*** é onde fica armazenado todo o histórico de alterações que no fim resulta no «*World State*». As transações são colocadas dentro de blocos que posteriormente são adicionados ao *Blockchain*. Transações essas que uma vez escritas não pode ser modificadas nem removidas. Isto é chamado de imutabilidade.

[4] Endorsement: https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html

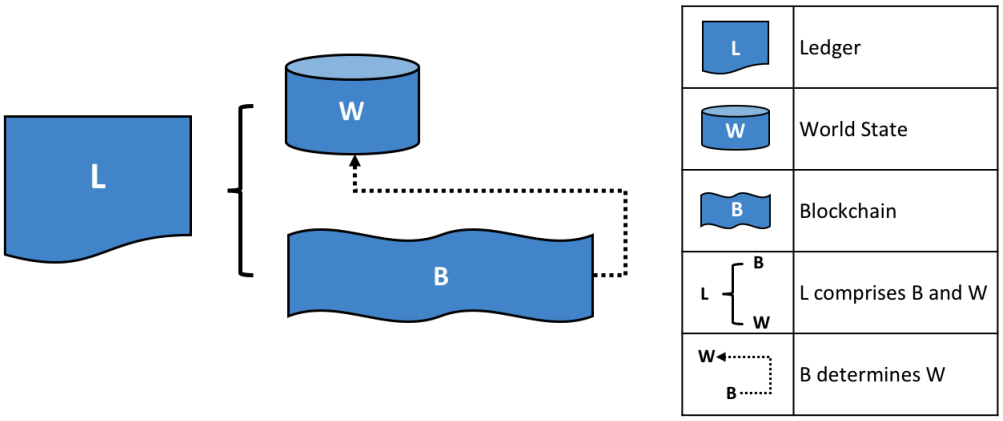

Figura 3.4: Ledger, World State e Blockchain

(Imagem retirada da documentação do Hyperledger Fabric https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html)

A rede mantém varias cópias de um «*Ledger*». Cópias essas mantidas pelos diferentes «*Peers*». Essas cópias são mantidas consistentes entre si num processo chamado de «*Consensus*».

3.2.8 Chaincode

Antes de se puder efetuar qualquer transação dentro da rede de *Blockchain* é importante definir um conjunto de contratos que tenham em conta os termos, os dados, as regras e os processos que definem transações que serão trocados entre as organizações. O «*Chaincode*» está representado na figura 3.1 como «S». Em suma pode dizer-se que um «*Chaincode*» define as regras entre as diferentes organizações recorrendo a código executável. As aplicações podem invocar as funções do «*Chaincode*» para gerarem transações que serão armazenadas no «*Ledger*». Tipicamente os «*Chaincodes*» são instalados nos «*Peers*» de uma organização. Nem todos os «*Peers*» precisam de ter o «*Chaincode*» instalado.

3.2.9 Peer

Um «Peer» é uma entidade da rede que mantém uma cópia do «Ledger» e é onde são executados os «Chaincodes», para realizar operações de leitura e escrita no «Ledger». O «Peer» está representado na figura 3.1 por «P». São mantidos e são propriedade das organizações.

Estes têm uma identidade assinada por um certificado digital tal como outros elemen-

tos da rede. Certificado esse emitido pelo «CA» afeto à organização a qual está inserido, através do «MSP».

Dentro de uma organização há vários tipos de «Peers». Estes são categorizados consoante a tarefa que desempenham dentro da organização.

Leader Peers

Este «Peer» recebe os novos blocos vindos dos «Orderers» e partilha-os com os restantes «Peers». Para o fazer recorre ao protocolo «Gossip»[5].

Anchor Peers (Peers Âncora)

Apenas os «Peers» âncora são conhecidos por outras organizações. Estes comunicam com os «Peers» âncora de outras organizações recorrendo ao protocolo Gossip.

Endorsing Peers

Quando uma rede é construída configura-se quais os «Peers» que irão realizar «Endorsement» às transações. Estas regras são chamadas de políticas de «Endorsement». Esta política define uma lista de «Peers» responsáveis. Tipicamente o «Chaincode» só é instalado nestes «Peers». Por omissão todos os «Peers» da rede são «Endorsing Peers».

3.2.10 Channel (Canal)

Um canal é uma espécie de rede de *Blockchain* privada. Permite o isolamento e a confidencialidade dos dados e das comunicações. É permitido que múltiplas organizações trabalhem com vários canais em simultâneo. Um canal é composto por dois sub-canais. Canal do sistema de ordenação e o canal de aplicação.

Canal do sistema de ordenação

Este define um conjunto de nós de ordenação que irão formar o serviço de ordenação. Define um conjunto de organizações que serão administradoras do serviço de ordenação. Este inclui também organizações que são membros do «Consortium» do Blockchain. O «Consortium»[6] é composto por um conjunto de organizações que pertencem ao Canal do sistema. Os membros do «Consortium» tem a capacidade de criar novos canais e adicionar organizações do «Consortium» ao canal.

[5]Gossip: https://hyperledger-fabric.readthedocs.io/en/latest/gossip.html
[6]Consortium: https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html

Canal de aplicação

Este é o canal que irá conter as organizações dos «Peers». O Canal do sistema é usado como *template* para criação do canal de aplicação. Os nós de ordenação definidos no canal de sistema tornam-se as unidades de consenso do novo canal.

3.2.11 Comunicação entre nós da rede

Todas as comunicações entre os nós da rede usam «TLS» (*Transport Layer Security*). Da mesma forma que os nós «*CAs*» serão responsáveis por emitir os certificados que servirão para os nós se identificarem entre si, a entidade «*TLS CA*» será responsável por emitir os certificados que garantirão a segurança nas comunicações entres os nós da rede.

3.2.12 Como se processa uma transação?

Chamamos de transação à troca de dados que existe entre um cliente e a rede de Blockchain. As transações podem ser de dois tipos, consultas (queries) e atualizações (updates). Todas as transações que envolvem atualização da rede de Blockchain precisam do de ser aprovadas por todos os «Peers» envolvidos, num processo chamado de «Consensus». Na figura 3.5 é apresentado de forma sucinta uma transação.

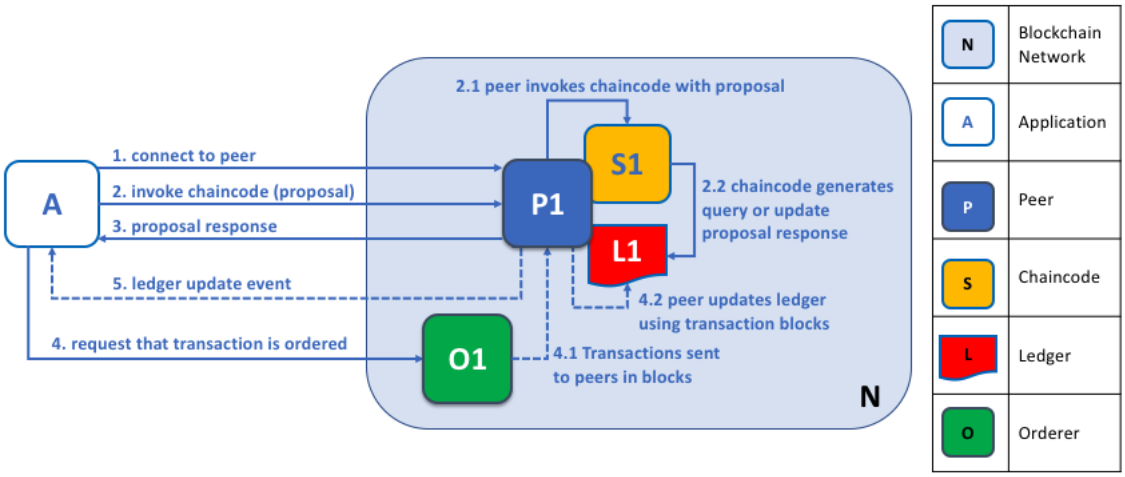

Figura 3.5: Fluxograma de uma transação

(Imagem retirada da documentação do Hyperledger Fabric https://hyperledger-fabric.readthedocs.io/en/release-2.2/peers/peers.html)

- Etapa 1: Proposta

A aplicação «A» presente na figura 3.5 liga-se ao canal, prepara e envia uma proposta de transação. Proposta que vai assinada digitalmente pelo utilizador da aplicação.

- Etapa 2: Peers recebem a transação

O «Peer» «P1» e todos os outros que receberem a transação são responsáveis pela sua validação. Estes verificam a sua formatação e se é a primeira vez que é enviada. Confirmam a assinatura com base no «MSP» da organização. Comprovam que o cliente da aplicação «A» tem autorização para realizar a operação em questão. A autorização é validada através da consulta das políticas de escrita do canal.

- Etapa 2.1: Invocação do Chaincode

Os «Peers» responsáveis pela transação vão nesta etapa invocar o seu «Chaincode» «S1» com a proposta de transação recebida. O «Chaincode» executa uma simulação contra a cópia do «Ledger» «L1».

- Etapa 3: Produção de um resultado

Se a transação for uma consulta a transação termina. O «Peer» retorna uma resposta à proposta de transação com o resultado da proposta de transação. Se a proposta de transação tiver como finalidade efetuar alterações ao Blockchain então serão retornados conjuntos de chave/valor representativos da proposta efectuada. Nesta fase nenhuma atualização é realizada no «Ledger». Todos os valores retornados são acompanhados da assinatura digital do «Peer» que executou e validou a proposta de transação, bem como de todos os «Peers» da rede que validaram a proposta. Esta acção é chamada de «Endorsement» e usa também o componente «MSP» para o efeito. Todo este processo de validação culmina com o consenso de todos os «Peers». Esta acção é chamada de «Consensus».

- Etapa 4: Invocação dos Orderers

Depois de A receber de volta o resultado da transação se esta for para modificar o Blockchain procede-se à invocação dos «Orderers». A aplicação «A» vai invocar «O1» e outros «Orderers» que existam no canal. Os «Orderers» não precisam de validar a transação. A transação neste ponto é composta por, ID do canal, o conjunto de chave/valor e a assinatura digital de todos os «Peers» que processaram a transação.

- Etapa 4.1: Ordenação e criação de blocos

Os «Orderers» envolvidos nesta acção irão agora ordenar a transação. Vão também criar os blocos representativos desta transação. Os blocos da transação são entregues a todos os «Peers» do Canal.

- Etapa 4.2: atualização do Ledger

Os «Peers» atualizam a sua cópia do «Ledger» com os blocos recebidos. Cada «Peer» antes de adicionar os blocos recebidos ao seu «Ledger», volta a validar se o «Endorsement» foi realizado corretamente, segundo as políticas do canal. Finalizada essa acção a aplicação é notificada do estado da transação.

3.3 Processo eleitoral

Com base nesta informação e na capacidade destes sistema é possível criar um processo eleitoral fiável e seguro. Foi tido como modelo uma eleição presidencial. O foco foi mantido nos votos recolhidos nas freguesias do concelho de Ourique, do distrito de Beja. Este processo organizativo é exemplo do que será aplicado a todos os concelhos de Portugal. O sistema de votação é composto por uma aplicação. Usada pelo votante para exercer o seu direito de voto e comunicar a intenção de voto à rede Blockchain. A utilização da aplicação pode ser feita através de um qualquer Smartphone, computador, e em qualquer lugar. Caso o votante não se sinta confortável ou não tenha habilitações para usar a aplicação, pode fazê-lo nas estações de vóto presentes nos locais de voto tradicionais.

3.3.1 Estrutura Aplicacional

O processo eleitoral aqui apresentado está dividido em três grandes áreas: O processo de autenticação; A aplicação usada pelo votante para interagir com o sistema e a rede de Blockchain que garante a validade e a segurança de todo o processo eleitoral.

- **Autenticação**

A autenticação é realizada por meio de um nome de utilizador e uma senha.

- **Aplicação**

A aplicação que é utilizada pelos votantes tem duas componentes. A componente UI/UX denominado de cliente que permite ao utilizador interagir com a rede de Blockchain. E uma API que é responsável por receber os pedidos vindos do cliente. A API fornece código com a capacidade de realizar as seguintes ações:

1. Autenticar votante
2. Listar eleições
3. Submeter votos
4. Consultar resultados
5. Criar eleição *

6. Iniciar eleição *
7. Terminar Eleição *

As ações com * apenas podem ser realizadas por uma entidade administrativa do sistema. Não estão disponíveis no cliente.

Depois de com o recurso à API se receber os pedidos do cliente em cada uma das suas ações disponíveis, os mesmos são encaminhados para a rede de Blockchain. Esse encaminhamento é feito depois de se usar o código da API para, se autenticar perante a rede.

- **Rede de Blockchain**

A rede de Blockchain é baseada na estrutura da rede padrão mencionada no sub-capítulo 3.2 «O que é o Hyperledger Fabric?». Os «MSPs» da rede contêm a lista de todas a entidades que podem interagir com a rede. Essa lista é usada em conjunto com o mecanismo de autenticação para aferir se um votante pode ou não submeter um voto. Todos os pedidos codificados na API para interacção com o «Ledger» passam pela execução das funções definidas no «Chaincode».

O «Chaincode» como referido no sub-capítulo anterior contém as regras e os processos necessários para que uma transação seja valida. As ações que estão disponíveis são:

1. Iniciar Sessão
2. Terminar Sessão
3. Listar eleições
4. Criar eleições
5. Iniciar eleição
6. Consultar resultados
7. Terminar eleição
8. Submeter voto

3.3.2 Divisão organizativa

a unidade administrativa mais pequena, foi tido em conta para facilitar a construção da rede de Blockchain. Esta é a freguesia. Cada organização criada representa uma freguesia de Portugal. Na figura 3.6 estão representadas a laranja as organizações que farão parte do canal (a Azul). O canal a azul é representativo dos atos eleitorais. Estão representados a verde os «Orderers». Desta forma o concelho de Ourique será composto por 4 organizações, Ourique, Santana da Serra, Panóias e Conceição e Garvão e Santa Luzia. Todas estas organizações representam freguesias do concelho.

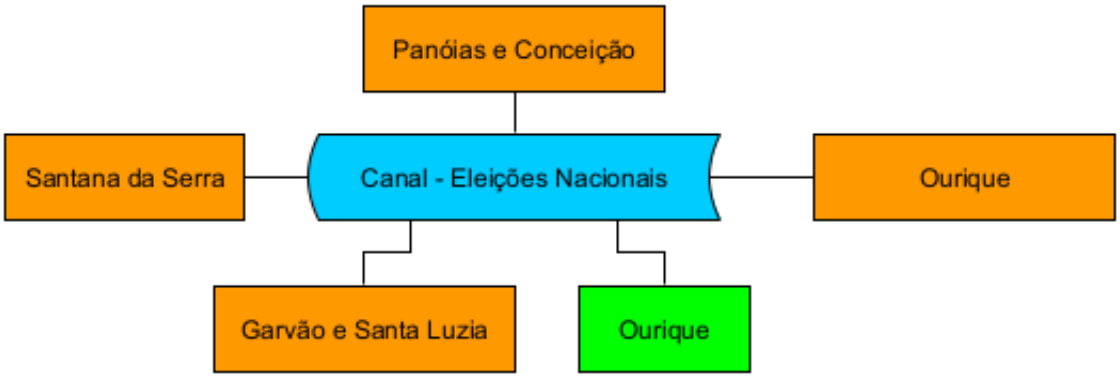

Figura 3.6: Estrutura Organizativa da rede de Blockchain.

3.3.3 Processo de votação

O processo de votação é semelhante ao que ocorre hoje em dia de forma presencial. Cada freguesia tem no seu «MSP» o registo de todos os eleitores que ali votarão. Cada freguesia é composta por um número de «Peers» que se designam de Estações de voto, em função do número de votantes da freguesia. É assim garantido o consenso necessário na avaliação de voto submetido.

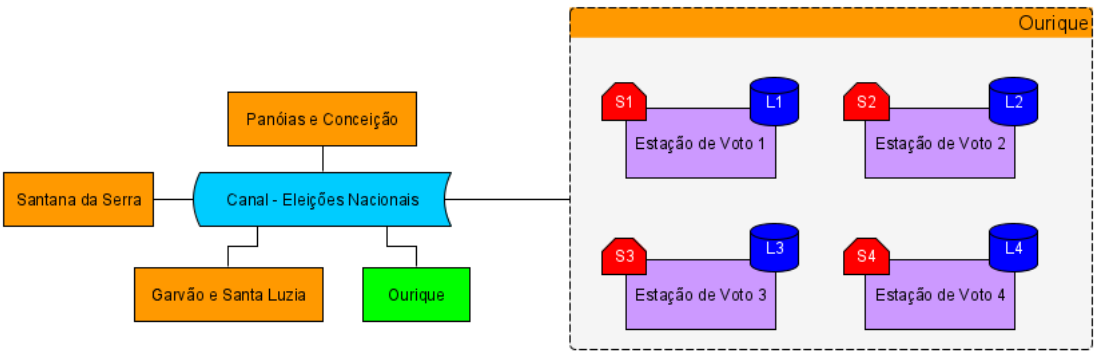

Figura 3.7: Estrutura Organizativa com representação dos Peers, Ledger e Chaincode.

Na figura 3.7 apresentam-se os «Peers» da freguesia Ourique. Todos possuem uma cópia do «Ledger». Tem instalado o «Chaincode» que será invocado pela Aplicação sempre que se quiser interagir com o «Ledger».

Acção - Autenticação do votante

Para o processo de autenticação é usado um nome de utilizador e senha.

- Etapa 1: Utiliza-se um nome de utilizador e uma senha. Depois de validadas as

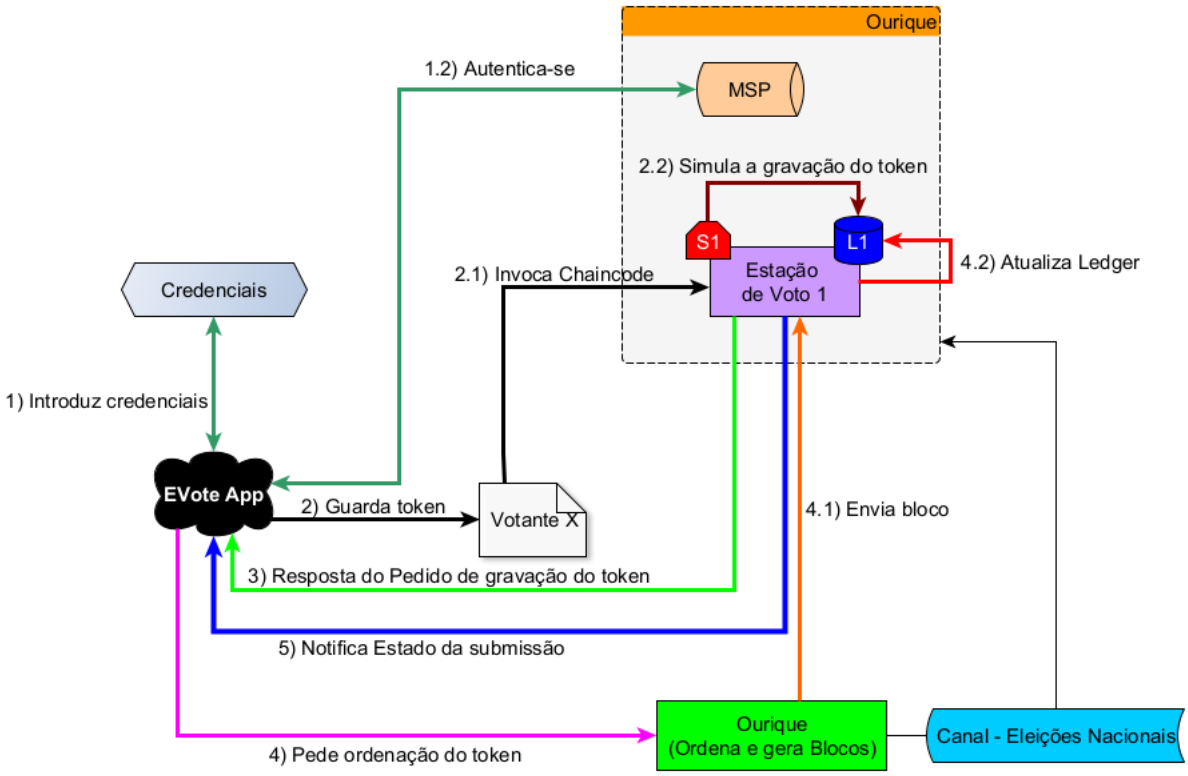

Figura 3.8: Processo de autenticação.

credenciais é realizada a autenticação junto dos «MSPs» do canal. Dos quais faz parte o «MSP» da Organização Freguesia de Ourique.

- Etapa 2: É gerado um «token» que é armazenado no «Ledger» juntamente com informação sobre o votante. Esse «token» é usado para validar cada pedido de **leitura de dados** que ocorra daí em diante na aplicação.

- Etapa 2.1: Com a informação enviada da aplicação é invocado o «Chaincode». Este tem uma função de «Login» que é responsável por iniciar o processo de gravação do «token» na rede.

- Etapa 2.2: A função «Login» do «Chaincode» simula a gravação do «token» associado ao votante.

- Etapa 3: Se todos as estações de voto (Peers) que receberem este pedido o considerarem correto este é retornado para a aplicação. O retorno será acompanhado de informação indicativa de que a operação é valida.

- Etapa 4: Inicia-se o processo de pedido de ordenação do «token» aos «Orderers». Este processo culmina com a atualização do Blockchain (Ledger). Os «Orderers»

ordenam e criam os blocos necessários para que seja possível atualizar a informação no Blockchain.

- Etapa 4.1: Estando o «token» convertido em blocos estes são distribuídos por todas as estações de voto (Broadcast).
- Etapa 4.2: Antes das estações de voto atualizarem o seu «Ledger» toda a informação é validada uma ultima vez. Por fim os blocos são adicionados ao «Ledger».
- Etapa 5: Correndo tudo bem, uma notificação de sucesso é entregue à aplicação. O «token» é utilizado daí em diante durante um determinado período de tempo.

Acção - Listar eleições

Através da aplicação o votante pode listar todas as eleições às quais seja elegível para votar.

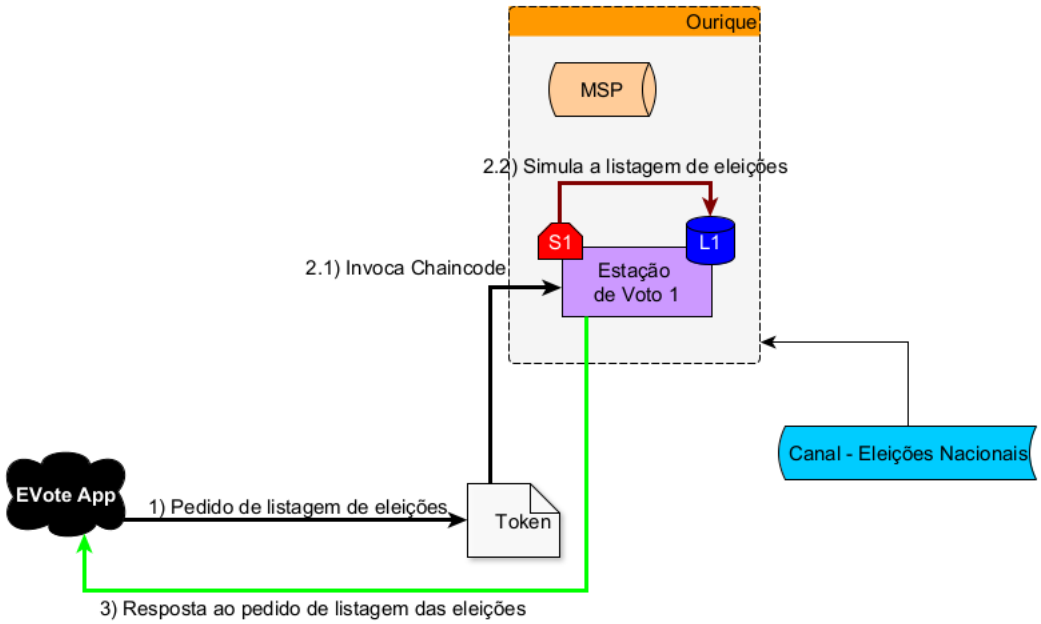

Figura 3.9: Processo de listagem das eleições.

- Etapa 1: Utilizando o «token» gerado anteriormente é feito um pedido de lista de eleições
- Etapa 2.1: Com a informação enviada da aplicação é invocado o «Chaincode». Este têm uma função «ListElections» que é responsável pedir ao «Ledger» todas as eleições em que o votante seja elegível.

- Etapa 2.2: A função «ListElections» do «Chaincode» simula a obtenção das eleições no «Ledger».
- Etapa 3: Sendo uma operação de leitura não é necessário o «Endorsement» nem o consenso geral de todos os «Peers». Sendo retornada a informação presente no «Ledger» do «Peer» que recebeu a invocação.

Acção - Consultar resultados

A consulta de resultados ocorre depois da eleição estar terminada. Esta acção desenrola-se da mesma forma que a listagem de eleições. A consulta de resultados é uma operação de leitura. Não necessita de «Endorsement», nem consenso da rede.

Acção - Submeter um voto

Através da aplicação é possível submeter o voto num candidato. O primeiro passo é a autenticação do utilizador da aplicação.

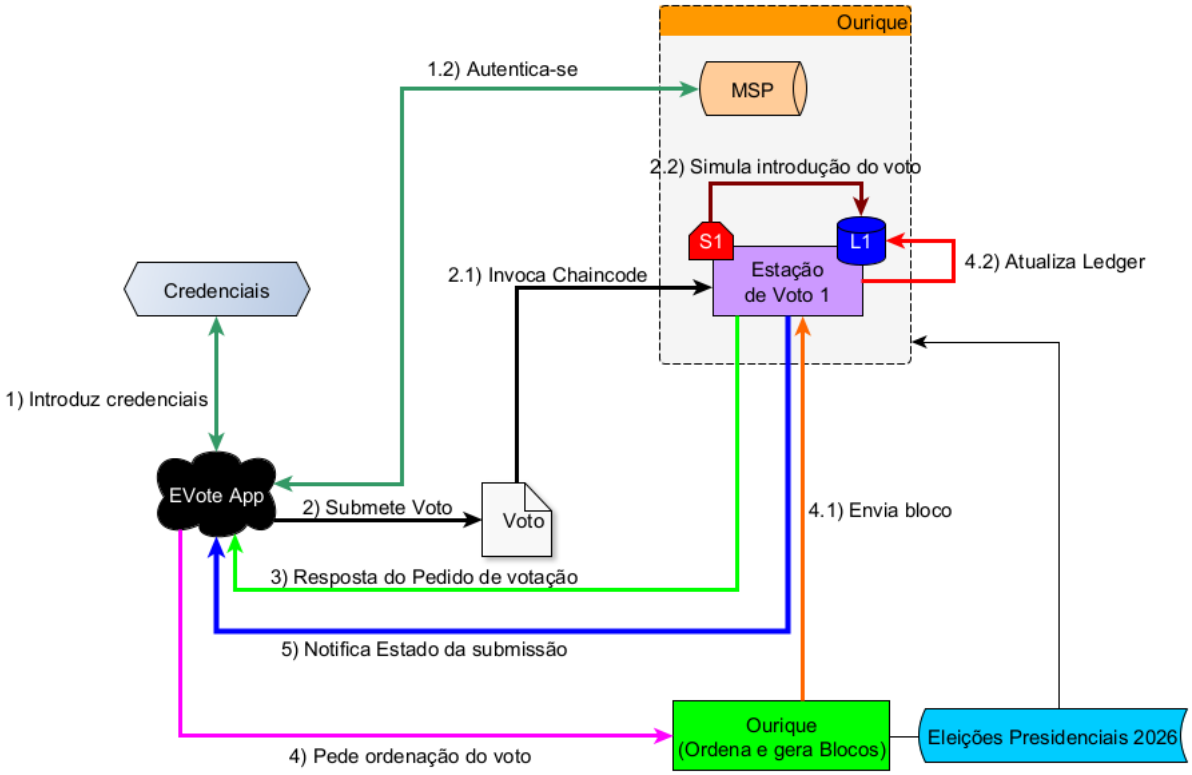

Figura 3.10: Processo de submissão de voto.

- Etapa 1: Utilizando o «token» é realizada a autenticação junto dos «MSPs» do canal. Dos quais faz parte o «MSP» da Organização Freguesia de Ourique.

- Etapa 2: Depois de seleccionada a eleição e o candidato, submete-se o voto. O voto é composto por informação de quem votou e qual a intenção de voto. Em momento algum essa informação é armazenada em conjunto.
- Etapa 2.1: Com a informação enviada da aplicação é invocado o «Chaincode». Este tem uma função «CastVote» que é responsável por iniciar o processo de submissão do voto na rede.
- Etapa 2.2: A função «CastVote» do «Chaincode» simula o incremento do voto no candidato seleccionado. É simulada a atualização da informação indicativa de que o utilizador já votou.
- Etapa 3: Se todos as estações de voto (Peers) que receberem o voto o considerarem correto este é retornado para a aplicação. O retorno é acompanhado de informação indicativa de que a operação é valida.
- Etapa 4: Inicia-se o processo de pedido de ordenação do voto aos «Orderers». Este processo culmina com a atualização do Blockchain (Ledger). Os «Orderers» ordenam e criam os blocos necessários para que seja possível atualizar a informação no Blockchain.
- Etapa 4.1: Estando os votos convertidos em blocos estes são distribuídos por todas as estações de voto (Broadcast).
- Etapa 4.2: Antes das estações de voto atualizarem o seu «Ledger» toda a informação é validada uma ultima vez. Por fim os blocos são adicionados ao «Ledger».
- Etapa 5: Estando tudo correto uma notificação do sucesso é entregue à aplicação.

Acção - Criar eleição e Iniciar/Terminar eleição

Todas as ações que envolvem modificações no «Ledger» passam pelos mesmos passos referidos acima. Ao contrário do que acontece numa operação de leitura que pode ser feita apenas com a participação de um «Peer». Nas operações que envolvem modificações do «Ledger» é sempre necessário que haja o consenso por parte de toda a rede.

3.4 Conclusão

O facto de o processo de consenso ser diferente do praticado pelas criptomoeadas aumenta a sua performance. Tornando o sistema ideal para desenvolver um sistema de votação electrónica. Este têm capacidade de ser distribuído, escalável e fiável. Uma solução capaz de garantir a cobertura nacional em qualquer eleição. Tornando possível a votação em qualquer ponto do planeta que disponha de um acesso à Internet. Devido ao seu «Ledger»

distribuído a veracidade da eleição está assegurada. Um voto só é aceite se a maioria dos nós da rede (Peers) o validar, o que torna difícil qualquer tentativa de fraude eleitoral. O «MSP» tendo conhecimento de todos os eleitores apenas permitirá que eleitores com credenciais validas exerçam o seu direito de voto. Isto só é possível graças ao uso de certificados digitais que atestam que o eleitor é quem diz ser. Sendo todo o processo eleitoral assente na confiança entre os nós da rede e os atores que interagem com ela a sua performance aumenta. Deste modo teremos uma eleição mais fluida sem tempos de espera na submissão dos votos. Para a que haja uma interação entre o eleitor e o sistema a API faz a ponte entre a aplicação cliente e o «Chaincode» que será responsável por validar as transacções propostas pela API e gerar uma resposta que será usada posteriormente para que os votos sejam colocados em blocos e submetidos na rede. Esta é a receita seguida por todas as transações realizadas, havendo apenas a diferenciação consoante o tipo de transação que se quer realizar. Leitura não precisam do consenso da rede e apenas precisam de interagir com um «Peer», enquanto que submissões requerem o esforço de toda a rede.

Capítulo 4

Realização Experimental

4.1 Introdução

Após o projeto da rede é possível proceder à implementação da mesma. Na criação desta rede foi utilizada a ferramenta Minifabric. A criação de redes com o uso do Hyperledger Fabric requer o manuseio de muitos ficheiros, tornando-se num processo manual propenso a erros. O Minifabric vem automatizar esse processo. Recorrendo a um template de configuração é estruturada a rede para a eleição presidencial. Terminada a criação da rede é depois construída uma API. Esta API usa o SDK disponibilizado pelo Hyperledger Fabric para que seja possível comunicar com a rede de Blockchain. Por fim é criada uma interface gráfica que serve de elo de ligação entre o utilizador e a API que faz a comunicação com a rede de Blockchain.

Este capítulo descreve como é a implementação da rede na secção 4.2 «A rede de Blockchain». Na secção 4.3 «O Minifabric» explica o que é o Minifabric e como funciona. Nesta secção todas a etapas de criação da rede, e os scripts produzidos para o efeito são analisados. na secção 4.4 «Criação de uma API usando o SDK», discute-se como foi construída a API de ligação à rede. A finalizar é descrito como foi criada a aplicação que o utilizador usará para interagir com a rede na secção 4.5 «Criação da aplicação cliente». Termina com a secção 4.6 «Conclusão» onde são apresentadas as conclusões a retirar desta realização experimental.

4.2 A rede de Blockchain

A rede será constituída por 3 091 organizações que representam as freguesias existentes em Portugal. Por 308 serviços de ordenação (Orderers). Representam os concelhos de Portugal. Todas estas entidades estão ligadas a um canal - Eleições Nacionais. Para efeitos de prototipagem o foco foi no concelho de Ourique, distrito de Beja. Esta rede é para este efeito composta por 4 organizações, representativas das 4 freguesias do concelho, por um serviço de ordenação. Cada organização é composta por um «Peer», aqui chamado de

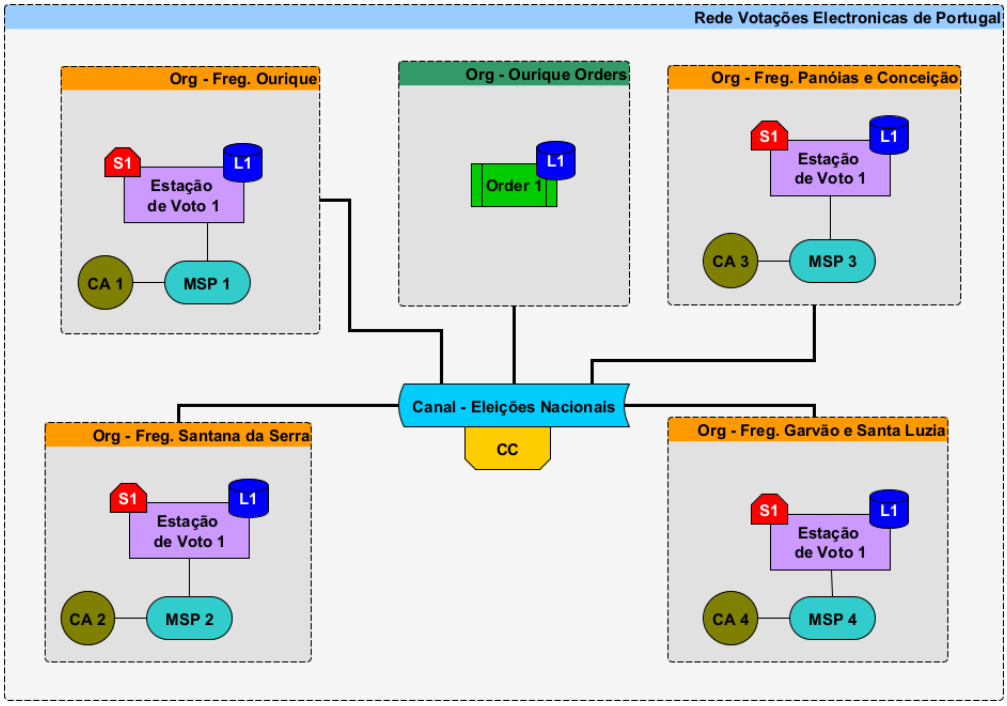

Figura 4.1: Processo de submissão de voto

«estação de de voto». Todos os «Peers» têm uma cópia do «Chaincode» instalada. Estes têm uma cópia do «Ledger». Cada organização tem um «CA» e um «MSP» que entre si vão fazer a gestão das entidades que interagem com as organizações respectivas. O canal tem um ficheiro de configuração. Estão descritas no ficheiro para além das configurações, as políticas pelo que o canal se rege.

No desenvolvimento deste protótipo é usado o Hyperledger Fabric na sua versão 2.3. Todos os nós da rede são representados por contentores Docker. A rede está configurada em servidores com distribuições Ubuntu Server.

4.3 O Minifabric

O processo de criação de uma rede de Blockchain utilizando o Hyperledger Fabric envolve muito trabalho manual. Este trabalho é muito complexo e moroso, propiciando a ocorrência de erros. Sabendo destas dificuldades membros da equipa que desenvolvem o Hyperledger Fabric na IBM, criaram esta ferramenta. O Minifabric[1] permite que haja uma abstração dos processos manuais. É uma ferramenta que permite configurar uma rede Hyperledger Fabric, expandir a rede, instalar e atualizar «Chaincodes», invocar tran-

[1]Minifabric: https://github.com/hyperledger-labs/minifabric

sações, inspecionar o «Ledger», alterar as configurações do canal entre outras coisas. Tudo isto de forma automatizada.

Atualmente o Minifabric permite realizar as seguintes acções:

- Lançar uma rede Hyperledger Fabric baseada num ficheiro de configuração;
- Eliminar uma rede criada;
- Operações a nível do canal tais como - Criar um canal, atualizar um canal, adicionar «Peers» a uma organização;
- Operações a nível do «Chaincode» tais como - instalar, aprovar, enviar, atualizar, inicializar, instanciar, invocar e realizar consultas;
- Consultar blocos e transações do «Ledger»;
- Gerar perfis de conexão e ficheiros de carteira (Wallet Files) para as linguagens go/node e python SDKs;
- Usar CouchDB;
- Usar Portainer.

Os scripts do Minifabric são construídos utilizando Ansible[2]. Esta é uma ferramenta de automação de tarefas de configuração. Torna, assim, a manutenção e criação de sistemas complexos e distribuídos mais simples.

4.3.1 Como utilizar

O Minifabric utiliza o Docker para ser executado num contentor e daí executar todas as suas tarefas. Em primeiro lugar é preciso obter o Minifabric e associá-lo a uma variável de ambiente com o seguinte comando:

```
1  mkdir -p ~/mywork && cd ~/mywork && curl -o minifab -sL https://
     tinyurl.com/yxa2q6yr && chmod +x minifab
2
```

Este comando também cria um diretório de onde serão executados todos os comandos do Minifabric. Neste diretório há um sub diretório «vars» onde serão criados scripts, templates, ficheiros intermédios entre outros.

No directorio /mywork deve ser adicionado o ficheiro em formato «YAML» de nome «spec». Este ficheiro tem o esqueleto da rede Hyperledger Fabric que será criada de forma automática pelo Minifabric

De seguida executa-se o comando que permite criar a rede com base nessa configuração definida no ficheiro «spec.yaml»

[2]Ansible: https://www.ansible.com/overview/how-ansible-works

```
minifab up -e 7778 -s couchdb -n fabcar -l node -v 1.0 -o orq.cne.com

```

Para além deste comando muitos outros podem ser executados para melhorar a rede. Por exemplo criar canais, adicionar organizações ou instalar «Chaincodes».

4.3.2 Criação da rede

A rede é criada pelo Minifabric com base no ficheiro spec.yaml colocado na raiz do diretório /mywork.

```
fabric:
  cas:
    - "ca1.orq.cne.com"
    - "ca1.sds.cne.com"
    - "ca1.pc.cne.com"
    - "ca1.gsl.cne.com"
  peers:
    - "peer1.orq.cne.com"
    - "peer1.sds.cne.com"
    - "peer1.pc.cne.com"
    - "peer1.gsl.cne.com"
  orderers:
    - "orderer1.cne.com"
  settings:
    ca:
      FABRIC_LOGGING_SPEC: ERROR
    peer:
      FABRIC_LOGGING_SPEC: ERROR
    orderer:
      FABRIC_LOGGING_SPEC: ERROR
  ### use go proxy when default go proxy is restricted in some of the
    regions.
  ### the default goproxy
  # goproxy: "https://proxy.golang.org,direct"
  ### the goproxy in China area
  # goproxy: "https://goproxy.cn,direct"
  ### set the endpoint address to override the automatically detected
    IP address
  # endpoint_address: 104.196.45.144
  ### set the docker network name to override the automatically
    generated name.
  netname: "mysite"
  ### set the extra optins for docker run command
  # container_options: "--restart=always --log-opt max-size=10m --log-
    opt max-file=3"
```

Listagem 4.1: Ficheiro de criação da rede

- **cas**

Aqui são definidos todos os «CAs» do canal por organização. Estes devem respeitar a nomenclatura na listagem 4.1.

- **peers**

Aqui são definidos todos os «Peers» do canal por organização. Estes devem respeitar a nomenclatura presente na listagem 4.1.

- **orderers**

Aqui são definidos todos os «Orderers» do canal. Estes devem respeitar a nomenclatura na listagem 4.1.

- **settings**

Aqui são definidos os níveis de logging para cada uma dos tipos de entidades. Por omissão é sempre DEBUG.

- **netname**

Nome do contentor docker e da rede de dockers que é responsável por correr os comandos gerados.

Dentro da diretoria /mywork é executado o comando de criação da rede referido na secção anterior.

Este comando executa uma série de playbooks. Estes fazem parte da estrutura de criação de scripts do Ansible.

4.3.3 Obter imagem Hyperledger Fabric

O comando de criação da rede Blockchain faz ao iniciar, a confirmação de que a imagem Docker do Hyperledger Fabric existe. Caso não exista é obtida dos repositórios do docker. É com esta imagem que todo o processo de criação é realizado.

4.3.4 Geração de certificados (certgen)

Esta é uma das etapas mais importantes de todo o processo de criação da rede de Blockchain. O Hyperledger Fabric cria redes do tipo «Permissioned». O consenso dos nós da rede neste tipo de rede é conseguido através da confiablidade na identidade dos ativos da rede. Essas identidades são compostas por certificados. Certificados que são usados também para assinar as transações para o uso do TLS na comunicação entre os nós. Para atingir este nível de confiabiidade são seguidos os seguintes passos:

1. Criação de ficheiro de configuração do material criptográfico.

   Com base no ficheiro de configuração «spec.yaml» colocado na raiz /mywork é gerado o ficheiro de configuração I.1 «crypto-config.yaml» que será usado para geração do material criptográfico.

   O ficheiro está dividido em duas partes.

   «OrdererOrgs» onde tem as configurações relativas aos «Orderers». O endereço «order1.cne.com» é decomposto em nome, domínio e hostname. O IP apresentado é da máquina onde estes serão instanciados.

   "PeersOrgs"tem uma estrutura semelhante à anterior com a adição de:

   - EnableNodeOUs - em caso de valor positivo, indica que este «Peer» tem capacidade de classificar identidades como sendo «clients» ou «Peers». Isto permite criar políticas tais como «Org.peer» ou «Org.client» para além das já existentes «Org.member» e «Org.admin».
   - CA - Informação que será usada para criar o «CA», que em conjunto com o «MSP» é responsável por gerir todas a entidades que integrem ou interajam com esta organização.

2. Criação do ficheiro de configuração «configtx.yaml».

   O «configtx.yaml» contém as configurações necessárias para criar um novo canal. Este ficheiro está na diretoria /mywork/vars. O ficheiro de configuração é composto pela seguinte estrutura:

   - Organizations - As organizações que podem tornar-se membros do canal. Cada organização referencia material criptográfico que é usado para construir o «MSP» do canal (conjunto dos «MSPs» das organizações).
   - Ordering service - Que nós de ordenação farão parte do serviço de ordenação da rede. O método de «consensus» que será usado para aceitar a ordem das transações.
   - Channel polcies - Define as políticas que irão gerir a forma como as organizações interagem com o canal e que organizações são precisas para provar atualizações ao canal.
   - Channel profiles - Os perfis usados para criar o bloco Génesis do sistema de ordenação do canal. Define quais os canais que irão ser usados pelas organizações dos «Peers».

3. Criação de certificados utilizando a biblioteca OpenSSL

Para cada organização são criadas duas chaves privadas usando comandos OpenSSL[3]. Uma ligada à autoridade certificadora «CA», que é colocada na diretoria /mywork/-vars/keyfiles/«organização» /ca com o nome priv_sk. Outra associada à autoridade certificadora da ligação entre os nós, «TLS CA», que é colocada na diretoria /mywork/vars/keyfiles/«categoria» /«organização» /tlsca com o nome priv_sk. Essas chaves são utilizadas para criar 2 certificados auto-assinados. Um para o «CA» e outro para o «TLS CA». Que são os certificados raiz da organização. Nas mesmas diretorias são colocados os certificados com a extensão de ficheiro $*.pem$. De seguida os certificados gerados são cópiados para a diretoria /mywork/vars/keyfiles/«categoria» /«organização» /msp dentro das pastas cacerts e tlscacerts.

Chegou o momento de se criarem os certificados para os outros ativos da organização. São criadas diretorias representativas dessas entidades dentro da diretoria da organização.

- É gerada uma chave privada para as ligações «TLS» desse ativo na sub-diretoria /tls.
- É gerada uma chave privada para essa entidade que será armazenada na sub-diretoria /msp/keystore.
- Gera-se um certificado para a entidade, assinado pelo certificado da organização. Com recurso à chave privada criada anteriormente.
- Gera-se um certificado para a ligação «TLS» nos mesmos moldes. Assinado pelo certificado «TLS» da organização. Com recurso à chave privada anteriormente criada.
- Os certificados são colocados nas sub-dierectorias /msp/signcerts e /tls respetivamente.

Por fim é gerado o bloco **Génesis**. São também gerados com o script listado em I.2 os ficheiros de pedido de adesão das organizações ao canal. Há informação detalhada acerca do material criptográfico a ser usado e as políticas a serem seguidas no script listado em I.3. Nomeadamente políticas de leitura, escrita, administração e «Endorsemente». Os ficheiros gerados são colocados na raiz da diretoria inicial /vars.

4.3.5 Colocar em contentores Docker os ativos da rede (netup)

Este comando é responsável por colocar todos os ativos da rede em contentores Docker. Os seguintes passos foram seguidos:

[3]Comandos OpenSSL: https://github.com/hyperledger-labs/minifabric/blob/main/playbooks/ops/certgen/orgkeygen.yaml

1. É criada uma rede Docker com o nome presente no ficheiro de configuração inicial «spec.yaml».

2. São criados os volumes Docker para todos os nós da rede (Peers, Orders, CA etc).

3. Com base em todas as chaves privadas geradas no processo de geração do material criptográfico, gera-se um ficheiro de configuração. Esse ficheiro contém todas as diretorias, onde se localizam as chaves privadas por ator da rede (Users, Orders etc).

4. Depois são criadas as variáveis de ambiente a serem usadas pelos contentores para todos os nós. Estas variáveis são armazenadas em ficheiros e usadas posteriormente quando criados os contentores.

5. É criado o contentor que irá conter a instância da BD *CouchDB*. Esta base de dados representa o «World State» dos dados.

6. São criados contentores para todos os nós da rede (Peers, Orders, CAs etc). Na criação desses contentores são aplicadas as variáveis de ambiente anteriormente criadas. O material criptográfico gerado na etapa anterior é também adicionado ao contentor.

4.3.6 Criação do canal (channelcreate)

Este comando irá criar o canal que representa as eleições nacionais. Tipicamente os canais são criados, construindo uma transação de criação de canal. De seguida essa transação é submetida para o serviço de ordenação. A transação de criação do canal especifica a configuração inicial do canal. Esta transação é também usada para o serviço de ordenação pedir a escrita do bloco Génesis.

Através do primeiro comando do script I.4 é criada a transação de criação do canal. O comando usa o perfil «OrgChannel» presente no ficheiro de configuração «configtx.yaml». A transação é armazenada no formato protobuf.

A transação é usada para posteriormente criar o canal através do segundo comando do script I.4. O comando usa o «Peer CLI» para enviar a transação de criação do canal para o serviço de ordenação que irá criar o canal. O «Peer CLI» é usado em conjunto com o ficheiro de configuração «core.yaml». O ficheiro I.5 «core.yaml» é obtido dos ficheiros de configuração do Hyperledger Fabric.

4.3.7 Juntar Peers ao canal (channeljoin)

Após a criação do canal chegou o momento de lhe juntar todos os «Peers». Para tal é necessário o bloco Génesis que foi gerado anteriormente. Para exemplo o comando I.6, ilustra a adição dos «Peers» da organização «Ourique» ao canal.

A identificação de qual o «Peer» a ser adicionado é feita através das variáveis de ambiente previamente definidas.

4.3.8 Define os Peers âncora.

Depois dos «Peers» serem adicionados ao canal as organizações devem definir pelo menos um dos seus «Peers» como «Peer» âncora. Os «Peers» âncora são usados pelo protocolo «Gossip» para garantir que os «Peers» em diferentes organizações tem conhecimento uns dos outros.

Para definir os «Peers» âncora e atualizar a configuração do canal usa-se a ferramenta «configtxlator». Para além da ferramenta «configtxtlator» é necessária a utilização da ferramenta «jq»[4]. São precisos seguir alguns passos, como os apresentados no script I.7 associados à organização freguesia de Ourique («orq»):

1. Obter os blocos de configuração mais recentes do canal;
2. Usar a ferramenta «configtxlator» e a «jq» e transformar o bloco de configuração obtido em JSON. O JSON produzido será usado como base para o processo de atualização do canal;
3. Atualiza-se o ficheiro de configuração JSON com informação do «Peer» âncora a ser adicionado ao canal;
4. Depois de atualizado o ficheiro de configuração JSON, o mesmo volta a ser convertido para o formato «Protobuf»;
5. Determinam-se as diferenças entre a configuração atual e a modificação realizada;
6. Coloca-se as diferenças em JSON;
7. Adicionam-se as diferenças a um bloco que será usado para produzir uma transação que atualizará o canal;
8. Transforma-se o bloco criado em «Protobuf»;
9. Um dos administradores da organização assinará a atualização e a mesma é submetida.

4.3.9 Criação de perfis de ligação (profilegen)

Um perfil de ligação descreve um conjunto de componentes. Esses componentes incluem «Peers», «Orderers» e autoridades certificadoras da rede de Blockchain recentemente criada. Contém também informações do canal e das organizações. Os perfis aqui gerados

[4]jq: https://stedolan.github.io/jq/

são depois usados pela aplicação cliente para que o utilizador possa interagir com a rede de Blockchain.

Para além destas configurações é criada também uma carteira por organização. Esta carteira será responsável por armazenar o material criptográfico no padrão X.509, que identifica os utilizadores que podem interagir com a rede em combinação com um «MSP».

4.3.10 Criação do Chaincode

O «Chaincode» implementa uma interface que permite a recepção de transações vindas dos clientes. Este implementa a lógica de negócio acordada por todos os membros da rede. O «Chaincode» desenvolvido para receber as transações da aplicação cliente está construído em NodeJS. Nele estão implementadas as seguintes funções:

Função - createElection

Esta função recebe o JSON de uma eleição em argumento como no código JSON 4.11 e cria a mesma.

Função - joinVotersToElection

Esta função recebe como argumento o Id de uma eleição. Obtém todos os votantes registado no «Ledger». Verifica quais os que são elegíveis para participar na eleição indicada. Regista a elegibilidade dos votantes filtrados no «Ledger».

Função - joinBallotsToElection

Esta função recebe a lista de candidatos à eleição e adiciona-os à dita eleição.

Função - closeElection

Esta função recebe o Id da eleição a ser terminada.

Função - openElection

Esta função recebe o Id da eleição a ser iniciada. O início da eleição vai também depender das datas de início e fim definidas quando a eleição foi criada.

Função - getBallot

Esta função recebe o Id de uma eleição e retorna os candidatos da mesma.

Função - createVoter

Esta função recebe informações sobre um votante (Nome, país, etc) e insere no «Ledger»

Função - createUser

Esta função cria um utilizador e associa o Id do votante previamente recebido como argumento.

Função - login

Esta função recebe um «token» de login e adiciona-o ao utilizador em questão. Juntamente com um tempo de expiração.

Função - loginStatus

Verifica o estado do utilizar. Se a sessão ainda continua válida.

O «Chaincode» é desenvolvido utilizando a biblioteca fabric-contract-api. Esta disponibiliza uma interface de alto nível, que permite abstração na interação com o «Ledger».

Estão criados contratos representativos da eleição, votante, utilizador e do boletim de voto. Os ficheiros com o código do «Chaincode»pode ser encontrado no apêndice III.

Em 4.2 está representada a função do «Chaincode» responsável por criar uma eleição. Em termos estruturais todas as outras funções vão ter a mesma estrutura.

```
async createElection(ctx, args) {

    args = JSON.parse(args);

    let response = {};

    let electionExists = await this.myAssetExists(ctx, args.electionId
    );

    if (electionExists) {
        response.error = `An election with ${args.electionId} id
    already exist`;
        return response;
    }

    let electionStartDate = args.electionStartDate;
    let electionEndDate = args.electionEndDate;

    //create the election
    let election = await new Election(args.electionId, args.
    electionName, args.electionCountry,
        args.electionYear, electionStartDate, electionEndDate);

    console.log("Election# " + election.electionId);

```

```
23      const strElection = JSON.stringify(election);
24      console.log(strElection);
25
26      response = await ctx.stub.putState(election.electionId, Buffer.
        from(strElection));
27
28      console.log("#################### Election generated successfully")
        ;
29
30      return response;
31  }
```

Listagem 4.2: Função de criação de eleição do chaincode

A função representada pelo código 4.2, utiliza a biblioteca fabric-contract-api para gerir todo o processo de interação com o «Ledger».

- A função recebe sempre como argumentos a variável «ctx». Esta representa o contexto gerado pela biblioteca e que permite a interação com o Blockchain. Os argumentos (args) recebidos da API estão em formato JSON;
- É feito uma conversão da «string» recebida para uma estrutura JSON;
- Verifica-se se a eleição existe através da chamada da função «myAssetExists». Passando em argumento o identificador da eleição. A função vai executar - ctx.stub.getState(myAssetId); - que tenta obter o ativo da rede com base numa chave;
- Caso não exista é criado um objeto «Election» com os dados recebidos. Por fim este é convertido novamente para «string»;
- Antes de ser colocado no Blockhain através da função «putState» a eleição em formato «string» é convertida para bytes. Todos os ativos das redes de Blockchain são armazenados numa combinação de chave-valor. Sendo que aqui a chave é o identificador da eleição e o valor a eleição convertida em bytes.

4.3.11 Instalar o Chaincode (ccinstall)

É importante que o código esteja na diretoria /vars/chaincode/(linguagem em que foi desenvolvido)/(nome do Chaincode) para se proceder à instalação do «Chaincode». Neste caso a linguagem foi NodeJS, logo a diretoria deverá designar-se de «node».

A instalação do «Chaincode» deve ser realizada em todos os «Peers» que executem e realizem «Endorsement» de transações.

Antes de realizar esta ação é necessário empacotar o «Chaincode» no formato tar.gz.

O pacote irá conter dois ficheiros. Um ficheiro de meta-dados e outro com os ficheiros do «Chaincode». O ficheiro de meta-dados especifica a linguagem de programação usada, a diretoria do «Chaincode» e o nome do mesmo.

Terminado o empacotamento deve ser realizada a instalação. Esta ação deve ser realizada por um «Peer» administrador. O «Peer» irá compilar o «Chaincode» depois de este estar instalado e retornar eventuais erros que ocorram.

Depois de instalado é retornado um identificador constituído pelo nome do «Chaincode» e um código «hash» do pacote. Este identificador será usado para associar o «Chaincode» instalado nos «Peers» com a definição do «Chaincode» aprovada pela organização. No script I.8 é possível ver este procedimento em detalhe para a organização da freguesia de Ourique («orq»).

4.3.12 Aprovação do Chaincode (ccaprove)

O «Chaincode» é gerido pela sua definição. A definição do «Chaincode» é um descritivo acerca do «Chaincode». A aprovação desse descritivo permite que os membros do canal aceitem o «Chaincode» antes deste ser usado num canal. A definição inclui os seguintes pontos:

- **Name**: O nome que as aplicações irão usar para invocar o «Chaincode»;
- **Version:** O número da versão do «Chaincode»;
- **Squence:** Número de vezes que um descritivo do «Chaincode» foi atualizado. Em suma este valor é incrementado sempre que há uma atualização do «Chaincode». Começando com 1;
- **Endorsement Policy:** Indica quais as organizações que precisam executar e validar o retorno de uma transação. Esta política por omissão usa «Endorsement» do canal, que está definida no seu ficheiro de configuração «configtx.yaml». Indica que a maioria das organizações de um canal devem realizar «Endorsement» de uma transação;
- **Collection Configuration:** Diretoria para um repositório de dados privados associados ao «Chaincode»;
- **ESCC/VSCC Plugins:** O nome do *plugin* personalizado de «Endorsement» e a validação usada pelo «Chaincode».
- **Initialization:** Aqui indica-se que função de inicialização deve ser chamada quando se der o processo de inicialização do «Chaincode». Esta função atua como uma

espécie de construtor, onde o código inicial pode ser executado. Esta função não tem necessariamente de ser invocada pelo mecanismo de inicialização. Depende do parâmetro –init-required usado no comando.

É imperativo que todos os membros que queiram usar o «Chaincode» aprovem a sua definição. Essa aprovação é realizada pela organização. A transação de aprovação gerada deve ser submetida a uma serviço de ordenação. Depois a definição é distribuída por todos os «Peers» da organização. Esta ação apenas pode ser executada pelo administrador da organização. A definição é armazenada numa colecção que está disponível para todos os «Peers» da organização. A aprovação é realizada com a execução do script I.9.

4.3.13 Enviar a definição do Chaincode ao canal (cccommit)

Depois da maior parte das organizações terem aprovado a definição do «Chaincode», uma organização pode envía-la ao canal. A transação de proposta de entrega da definição do «Chaincode» é primeiro enviada para os «Peers» do canal. Estes consultam a definição que foi aprovada pela sua organização e fazem o «Endorsement» da mesma. Depois a transação é entregue a um serviço de ordenação que posteriormente entrega a definição ao canal. Esta operação só pode ser realizada pelo administrador da organização.

O número de organizações que precisa de aprovar a definição do «Chaincode» é definido pela política «LifecycleEndorsement». Esta política está definida no ficheiro de configuração do canal «configtx.yaml». Este número pode ser diferente do necessário para aprovar um «Chaincode». Esta operação é realizada executando o script I.10.

4.3.14 Instanciação do Chaincode (ccinstantiate)

Este processo irá invocar o método de inicialização anteriormente criado. Para tal deve ser passado como argumento –isInit.

4.3.15 Serviço de Descoberta (discover)

O serviço de descoberta é o que permite que por meio da API e através do SDK se possa:

- Invocar o «Chaincode»;
- Submeter transações;
- Receber notificações sobre o estado das transações.

Sem este a aplicação não tem como saber, que os «Peers» têm os «Ledgers» atualizados e que os «Peers» têm o «Chaincode» instalado. Quais os que são responsáveis pelo «Endorsement» das transações.

O serviço de descoberta facilita a obtenção destas informações. Este faz com que os «Peers» computem a esta informação dinamicamente, tornando a informação disponível para o SDK usar.

4.4 Criação de uma API usando o SDK

Terminado o processo de criação da rede de Blockchain, que culminou com a instalação do «Chaincode», está na hora de criar uma API. Essa API é responsável por permitir a interação de uma aplicação com o «Chaincode». Para a construção da API foi usada a biblioteca «fabric-ca-client» do Fabric SDK, responsável pela geração de material criptográfico. Foi também usada a «fabric-network» responsável por gerir a ligação da API à rede de Blockchain. A API foi desenvolvida na linguagem NodeJS.

4.4.1 Ligação à rede Blockchain

Para a API se poder ligar à rede de Blockchain são requeridos quatro parâmetros. O ficheiro com os perfis de configuração, previamente criado. A identidade do utilizador, previamente conhecida pelo «MSP». O nome do canal ao qual será feita a ligação. O nome do «Chaincode» que vai ser invocado.

- O perfil de ligação é um ficheiro em JSON. Preparado para ser utilizado através do SDK NodeJS. Em 4.3 está o código que carrega o ficheiro. Com recurso à biblioteca «path» e «fs» do NodeJS

```
1 const ccpPath = path.join(process.cwd(), 'connection.json');
2 const ccpJSON = fs.readFileSync(ccpPath, 'utf8');
3 const ccp = JSON.parse(ccpJSON);
```

Listagem 4.3: Carregar o ficheiro configurações da ligação

- É necessário criar o objeto com acesso à carteira. Para tal é identificada a localização da carteira pretendida. É necessário também criar uma instância da «Gateway». Esta funcionará como ponte de acesso à rede. Em 4.4 o código que abre a carteira.

```
1 const gateway = new Gateway();
2
3 // Create a new file system based wallet for managing identities.
4 const walletPath = path.join(process.cwd(), '../profiles/vscode/
      wallets', 'orq.cne.com');
5 const wallet = await Wallets.newFileSystemWallet(walletPath);
```

Listagem 4.4: Carrega a carteira e instância a Gateway.

- Carregada a carteira é necessário tentar obter o utilizador a ser usado. Caso não exista o processo é abortado de imediato e uma mensagem de erro devolvida.

```
const userIdentity = await wallet.get(userName);
```

Listagem 4.5: Tentativa de obtenção do utilizador da carteira

- Através da «Gateway» invoca-se a função «connect» que recebe como argumento o ficheiro com a configuração da ligação. A carteira e o nome do utilizador. De forma opcional, deve ser utilizada a funcionalidade de descoberta. Feita a ligação é obtido o acesso ao canal, indicando o seu nome na função «getNetwork». Com o objeto «Network» representativo do canal é obtido o «Chaincode» através do seu nome com a função «getContract».

```
await gateway.connect(ccp, { wallet, identity: userName, discovery: {
    enabled: true, asLocalhost: false } });

// Connect to our local fabric
const network = await gateway.getNetwork('mychannel');


// Get the contract we have installed on the peer
const contract = await network.getContract('fabcar');
```

Listagem 4.6: Ligação ao canal e ao chaincode

Finalizado este processo é possível agora invocar as funções do «Chaincode» através do objeto «contract».

4.4.2 Estrutura da API

A API está construída em NodeJS. Está dividida em duas partes. Uma parte para ser utilizada pela aplicação cliente, representada no ficheiro II.1. Outra para ser invocada para o tratamento de dados administrativos, representada no ficheiro II.2. Ambas as APIs utilizam uma classe auxiliar responsável pela ligação à rede Blockchain, representada no ficheiro II.3. As duas APIs são responsáveis por construir os pedidos e trabalhar as respostas recebidas da invocação do «Chaincode».

4.4.3 Função - enrollAdmin - API Administrativa

Esta função tem como objetivo criar um utilizador administrador numa dada organização. Esta função apenas deve ser chamada uma vez para cada organização. Esta função recebe um pedido «POST» com o nome do utilizador administrador a ser criado. De seguida são dados os seguinte passos:

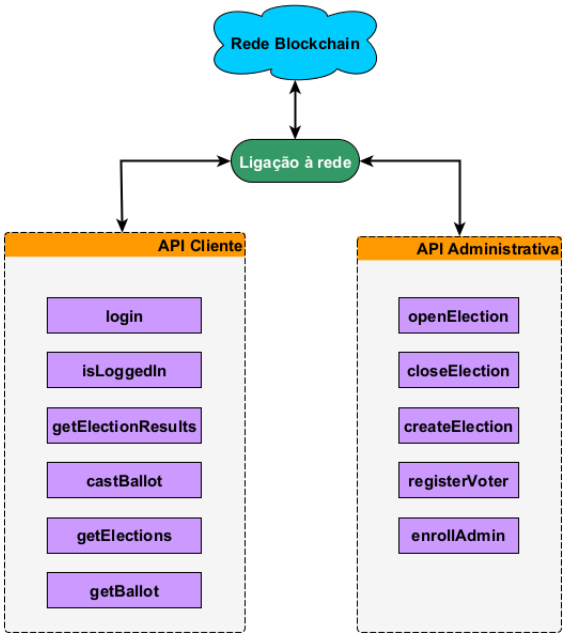

Figura 4.2: APIs, as suas funções e acesso à rede

- Chamada a função da classe auxiliar «enrollAdmin». Utilizando o ficheiro com as configurações dos perfis de ligação é obtido o «MSP ID» da organização em questão. É obtida também a informação do «CA» relativo àquela organização. Com a informação do «CA» é obtido o certificado da ligação «TLS». Com o recurso à biblioteca «FabricCA» do SDK, cria-se uma instância de «FabricCAService». Essa instância será responsável pelo registo do administrador no «MSP» da organização. Recebe como argumentos o URL do «CA». O certificado de «TLS» e o nome do «CA».

```
let mspId = ccp.organizations[orgConnection].mspid;

// Create a new CA client for interacting with the CA.
const caInfo = ccp.certificateAuthorities['ca1.org1.example.com'];
const caTLSCACerts = caInfo.tlsCACerts.pem;
const ca = new FabricCAServices(caInfo.url, { trustedRoots:
    caTLSCACerts, verify: false }, caInfo.caName);
```

Listagem 4.7: enrollAdmin - Criação do serviço CA para uso do recursos do CA

- O passo seguinte é o carregamento da carteira da organização, como realizado anteriormente. É verificado se o utilizador existe na carteira. Caso o utilizador não exista é seguido o processo de inscrição do administrador. Para isso é chamada a função «enroll» do «CA». Que recebe como argumento o nome do utilizador e a

sua senha. O resultado da função será usado para criar uma entidade do tipo X.509. Essa entidade é depois armazenada na carteira.

```
// Enroll the admin user, and import the new identity into the wallet.
const enrollment = await ca.enroll({ enrollmentID: appAdmin, enrollmentSecret: appAdminSecret });
const x509Identity = {
  credentials: {
    certificate: enrollment.certificate,
    privateKey: enrollment.key.toBytes(),
  },
  mspId: mspId,
  type: 'X.509',
};
await wallet.put(appAdmin, x509Identity);
```

Listagem 4.8: enrollAdmin -Inscrição do administrador e gravação da entidade resultante na carteira

Com esta inscrição do administrador no «MSP» da organização é agora possível registar utilizadores recorrendo ao administrador.

4.4.4 Função - registerVoter - API Administrativa

Esta função tem como objetivo realizar duas acções. Criar um utilizador, que será usado para fazer login na aplicação. Criar um votante que ficará associado ao utilizador. A função recebe um pedido «POST» com um ficheiro JSON como argumento.

```
{
  "username": "V8",
  "password": "V8pw",
  "registrarId": "Beja",
  "firstName": "Miguel",
  "lastName": "Costa",
  "country": "Portugal"
}
```

Listagem 4.9: registerVoter - Pedido de registo

O pedido contém, o primeiro e o último nome do votante. O nome de utilizador e a senha que serão usados para o login. A freguesia de registo e o país também fazem parte da informação enviada. Para registar o votante são seguidos os seguintes passos:

- Primeiramente é gerado um identificador único para o votante. Depois é chamada a função da classe auxiliar «registerUser». Esta função recebe como argumento o nome do utilizador. Utilizando o ficheiro com as configurações dos perfis de ligação é

obtido o «MSP ID» da organização em questão. É obtida também a informação do «CA» relativa àquela organização. Com o recurso à biblioteca «FabricCA» do SDK, cria-se uma instância de «FabricCAService». Essa instância será responsável pelo registo do utilizador no «MSP» da organização. Recebe como argumentos o URL do «CA». Todos os passos anteriores são semelhantes aos realizados na inscrição do administrador.

- O passo seguinte é o carregamento da carteira da organização, como realizado anteriormente. Verifica-se se o utilizador existe na carteira. Caso o utilizador não exista é seguido o processo e confirma-se se a carteira contém o utilizador administrador. Caso exista é obtida a informação acerca do administrador com base na carteira e na entidade obtida.

- Depois disso é invocada a função «register» do «CA». Esta recebe como argumentos, uma afiliação, o nome do utilizador, o tipo de registo e a informação sobre o administrador. É retornada uma chave que funcionará como uma senha. Esta chave é usada como argumento da função «enroll» do «CA». Esta recebe como argumento, o nome do utilizador e a chave obtida anteriormente. O último passo tal como no registo do administrador, é a inserção da entidade do utilizador na carteira.

```
1  const provider = wallet.getProviderRegistry().getProvider(
       adminIdentity.type);
2  const adminUser = await provider.getUserContext(adminIdentity,
       appAdmin);
3
4  // Register the user, enroll the user, and import the new identity
       into the wallet.
5  const secret = await ca.register({
6    affiliation: 'org1.department1',
7    enrollmentID: userName,
8    role: 'client'
9  }, adminUser);
10 const enrollment = await ca.enroll({
11   enrollmentID: userName,
12   enrollmentSecret: secret
13 });
14 const x509Identity = {
15   credentials: {
16  certificate: enrollment.certificate,
17  privateKey: enrollment.key.toBytes(),
18   },
19   mspId: mspId,
20   type: 'X.509',
21 };
22 await wallet.put(userName, x509Identity);
```

Listagem 4.10: registerVoter - Registo do utilizador

4.4.5 Função - createElection - API Administrativa

Esta função tem como objetivo criar a eleição. A eleição é criada com recurso ao administrador. A função recebe um pedido «POST». Esta recebe como parâmetro um objeto JSON com a estrutura da eleição abaixo:

```
{
    "electionId": "EP2021PT#20",
    "electionName": "2021 Eleições Presidenciais Portugal",
    "electionCountry": "Portugal",
    "electionYear": 2021,
    "electionStartDate": "2021-11-27",
    "electionEndDate": "2021-12-15",
    "votableItems": [
        {
            "votableId": "MJUPEP2021#2",
            "votableName": "Marcelo Rebelo de Sousa",
            "description": "O meu nome é Marcelo Rebelo de Sousa. E esta é a minha agenda",
            "electionId": "EP2021PT#20"
        },
        {
            "votableId": "MSATEP2021#2",
            "votableName": "Ana Gomes",
            "description": "O meu nome é Ana Gomes. E esta é a minha agenda.",
            "electionId": "EP2021PT#20"
        },
        {
            "votableId": "MMEREP2021#2",
            "votableName": "André Ventura",
            "description": "O meu nome é André Ventura. E esta é a minha agenda.",
            "electionId": "EP2021PT#20"
        },
        {
            "votableId": "MMAREP2021#2",
            "votableName": "João Ferreira",
            "description": "O meu nome é João Ferreira. E esta é a minha agenda.",
            "electionId": "EP2021PT#20"
        }
    ]
}
```

Listagem 4.11: registerVoter - Pedido de registo

A eleição para além da informação base será composta pela data de início e de fim e pelos candidatos. Depois são seguidos três passos. Todos eles usam a função auxiliar «invoke» da classe auxiliar. Esta função usa a ligação previamente criada para realizar a submissão de uma transação para a rede de Blockchain. Para tal é chamada a função «evaluateTransaction» ou «submitTransaction» da instância do contrato, consoante se queira fazer uma «Query» ou atualziar o Blockchain. Estas funções recebem como argumentos o nome da função a invocar no «Chaincode» e eventuais argumentos.

Os passos são os seguintes:

- Envia o pedido de criação de eleição para a função do «Chaincode», «createElection»;
- Invocada a função do «Chaincode», «joinVotersToElection» com o identificador da eleição. Esta função irá pegar em todos os votantes registados até ao momento. Filtrar quais os eleitores válidos a participar na eleição. No final é gerado um registo associando os votantes à eleição. Este registo será responsável por guardar a data em que o eleitor votou.
- A ultima ação é invocar a função do chaincode «joinBallotsToElection». Esta função recebe como argumento os candidatados presentes no objeto JSON.

4.4.6 Função - openElection - API Administrativa

Esta função tem como objetivo trocar o estado da eleição para ativo. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «openElection». Trocado o estado da eleição para ativo, se as datas da eleição estiverem válidas a mesma pode receber a participação dos votantes.

4.4.7 Função - closeElection - API Administrativa

Esta função tem como objetivo trocar o estado da eleição para inativo. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «closeElection». Trocado o estado da eleição para inativo, é possível aos votantes consultarem os resultados da eleição.

4.4.8 Função - login - API Cliente

Esta função tem como objetivo iniciar a sessão do utilizador na aplicação. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «login». É enviado como argumento o nome do utilizador, a senha, um «token» e

qual o tempo de vida da sessão. O «token» é gerado na API Cliente por limitações da rede de Blockchain. O «Chaincode» não pode gerar valores aleatórios sob pena de haver falhas no «Endorsement» da transação. Feita a alteração no utilizador, é devolvido o «token» para a aplicação cliente. Este deve ser utilizado em todas as chamadas da aplicação à API.

4.4.9 Função - isLoggedIn - API Cliente

Esta função tem como objetivo verificar se o token do utilizador ainda é válido. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «loginStatus». É enviado como argumento o «token» da sessão. Se o «token» estiver válido é retornada essa indicação para a aplicação.

4.4.10 Função - getElectionResults - API Cliente

Esta função tem como objetivo a obtenção dos resultados de uma eleição. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «getResultsByElection». É enviado como argumento o identificador da eleição. Caso esteja em condições de mostrar resultados, serão retornados os candidatos e a quantidade de votos obtidos.

4.4.11 Função - castBallot - API Cliente

Esta função tem como objetivo registar a opção de voto do votante. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «castVote». É enviado como argumento o «token» da sessão, o identificador da eleição, a data atual, o voto e o identificador do votante dentro na eleição. Caso todos os dados sejam validados pelo «Chaincode» o voto é submetido e incrementado.

4.4.12 Função - getElections - API Cliente

Esta função tem como objetivo listar todas a eleições que existem. É utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «queryByObjectType». É enviado como argumento o tipo de objeto a ser retornado («election»). Com a resposta recebida é criada uma lista de eleições que depois é retornada para aplicação.

4.4.13 Função - getBallot - API Cliente

Esta função tem como objetivo obter o boletim de voto de uma eleição. Para o fazer é utilizada a função «invoke» da classe auxiliar. É feita uma chamada à função do «Chaincode», «getBallot». Depois é enviado como argumento o «token» e o identificador da eleição. O boletim de voto obtido é retornado para a aplicação.

4.5 Criação da aplicação cliente

A aplicação cliente faz a ponte entre o utilizador e a rede de Blockchain. Esta é criada em VueJs. Está preparada para ser usada em dispositivos móveis. É composta por quatro áreas principais:

- A página de login, usada para autenticar o utilizador e guardar o «token» de sessão nos «Cookies» do navegador.
- A página de listagem das eleições disponíveis.
- A página de votação. Nesta aparece a lista de candidatos.
- A página de visualização dos resultados de uma eleição.

Todas as acções aqui realizadas são acedidas pela API cliente.

4.5.1 Página de login

Após o pedido de login é recebido um «token» que é armazenado nos «Cookies» do navegador. Todas as rotas da aplicação verificam se o «token» existe e está válido. Caso contrário são redirecionadas para a página de login. Caso o login falhe uma mensagem é mostrada e o utilizador deve tentar novamente. Na figura 4.3 está representada a página de login.

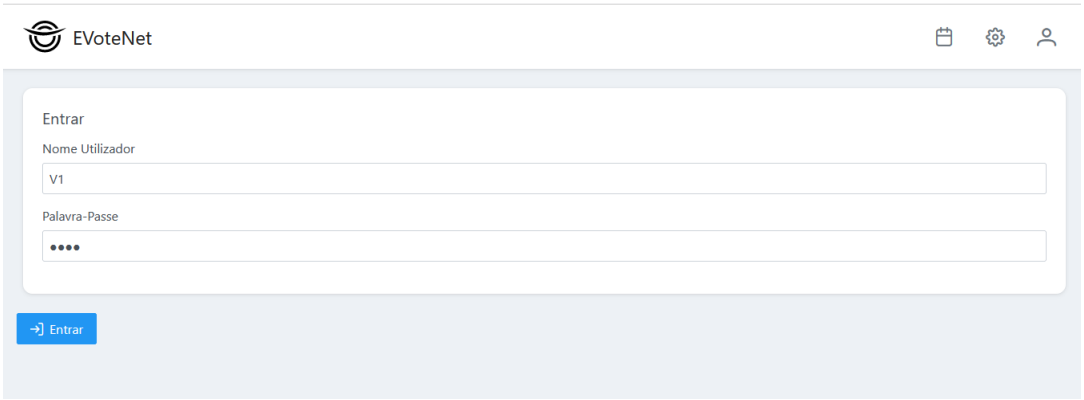

Figura 4.3: Página de Login

4.5.2 Página de listagem das eleições

Realizado o login, o utilizador é encaminhado para uma página que mostra uma lista de eleições. Esta lista de eleições apenas mostra eleições em que o utilizador como votante possa participar. Caso a eleição esteja a decorrer o utilizador pode votar. Caso esteja terminada é possível visualizar os resultados. Na figura 4.4 está representada a página que lista as eleições.

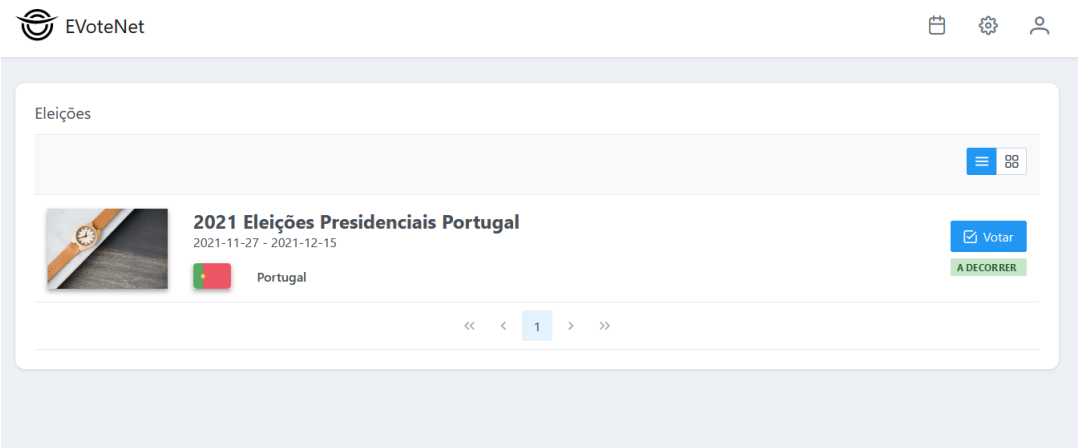

Figura 4.4: Página de listagem das eleições

4.5.3 Página de Votação

Caso a eleição esteja a decorrer o utilizador pode votar no botão «Votar». Na página de votação são exibidos todos os candidatos a votação na eleição em questão. Ao submeter o voto é questionado se tem a certeza. Caso afirmativo é reencaminhado para a página da listagem das eleições. Uma mensagem com o estado da submissão do voto é exibida. Na figura 4.5 está representada a página que mostra o boletim de voto.

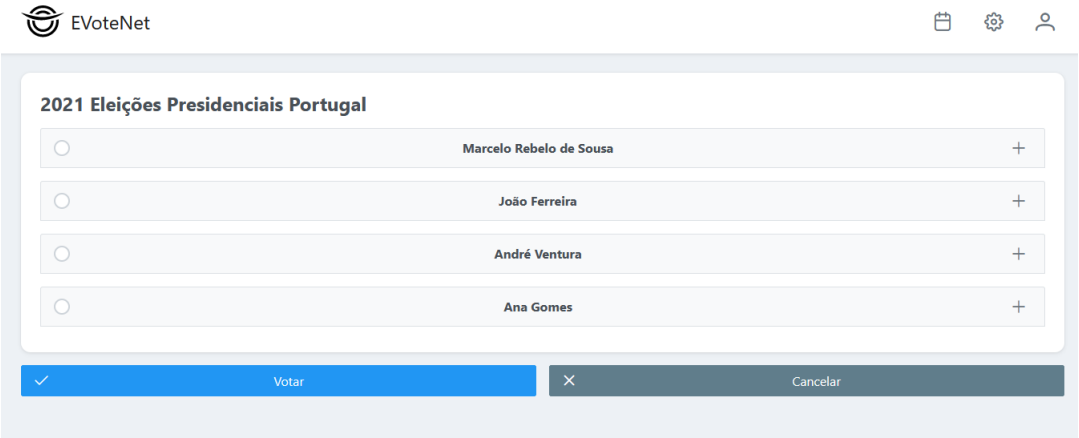

Figura 4.5: Página de votação

4.5.4 Página de visualização dos resultados

Quando a eleição está no seu estado «Terminado» é possível ao votante ver os resultado das eleições. O botão «Resultados» surge na página de listagem das eleições, nas eleições terminadas. Na figura 4.6 a página de listagem das eleições com a representação de uma eleição terminada.

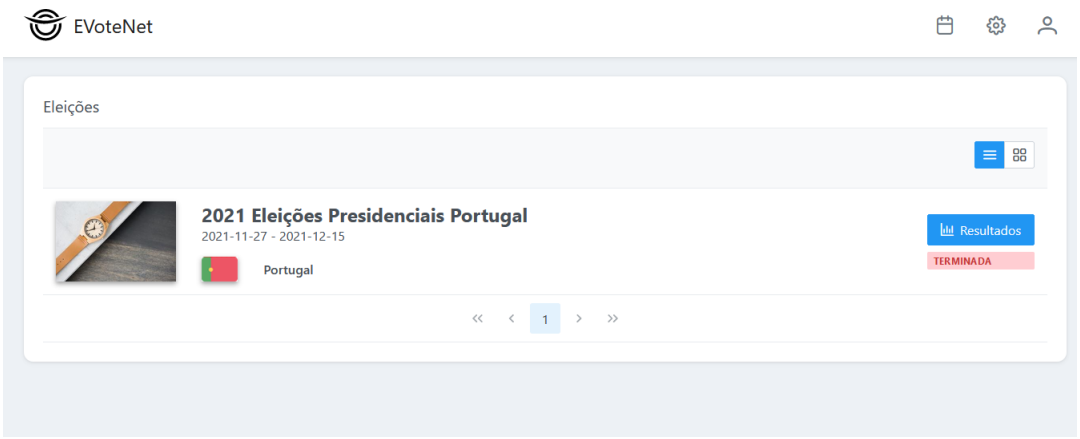

Figura 4.6: Página de listagem de eleições com possibilidade de ver resultados

Depois de ser encaminhado para a página dos resultados o votante tem acesso a um gráfico, que representa o resultado de cada candidato. Na figura 4.7 a página dos resultados.

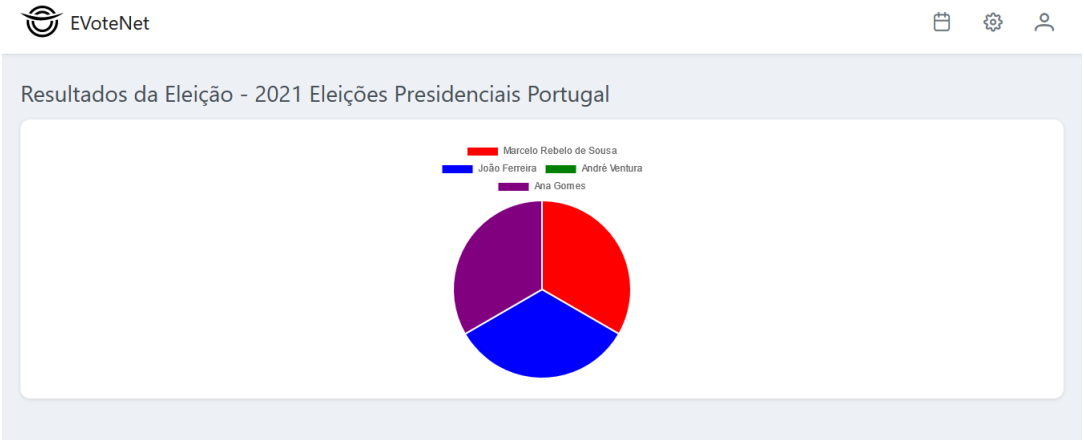

Figura 4.7: Página de resultados de uma eleição

4.6 Conclusão

A rede produzida pelo Hyperledger Fabric corresponde aos requisitos necessários para a realização de uma eleição segura e fiável. A utilização do Hyperledger Fabric requer uma curva de aprendizagem bastante acentuada. A documentação é extensa e detalhada contudo há aspectos de interpretação dúbia. Sendo esta uma Framework recente ainda não há muito debate nos principais fóruns da especialidade. Foi aqui que a ferramenta Minifabric, criada por colaboradores do Hyperledger Fabric interveio. Permitiu achatar a curva de aprendizagem e absorver melhor os conceitos que regem o Hyperledger Fabric.

Esta automatiza comandos de configuração, geração de scripts de configuração, criação de material criptográfico, a criação da rede (na qual se inclui o canal e todos os seus membros) e o empacotamento e instalação do «Chaincode». A criação do «Chaincode» foi facilitada pela possibilidade de usar uma linguagem de programação conhecida, o NodeJs. Ao contrário do que acontece com outras Frameworks do género que requerem o uso do Solidity. O seu desenvolvimento foi realizado através do uso de um SDK fornecido pelo Hyperledger Fabric que permite a interação com o «Ledger». Terminada a construção da rede, foi criada a API que faz a abstração da ligação à rede e permite que a aplicação cliente interaja com o Blockchain. Esta API foi construída em NodeJS com recurso ao SDK fornecido pelo Hyperldger Fabric. Com este SDK é possível criar proposta de transações que são depois de assinadas com recurso às entidades existentes na carteira, enviadas para a rede de Blockchain. As entidades armazenadas na carteira são parte fundamental de todo o mecanismo de confiança que rege uma rede criada pelo Hyperledger Farbic. São estas que asseguram que as transações recebidas pela rede são provenientes e criadas por entidade confiável. Por fim a aplicação cliente que foi criada usa uma das mais conhecidas Frameworks de front-end do mercado, o VueJs. Foi tido em conta que o seu uso teria que ser possível em qualquer dispositivo, estando assim abrangido um espectro maior de dispositivos.

Capítulo 5

Conclusão

O Blockchain tem uma capacidade aplicacional incrível, que está muito longe de esgotar toda a sua potencialidade. O caso disso são os mais diversos usos que tem hoje em dia. Desde o uso nas criptomoedas até ao uso particular dos sistemas de votação eletrónica. Já são alguns e muito diferentes entre si os sistemas de votação eletrónica baseados no Blockchain. Contudo a sua aceitação pelos governos de estados por todo o mundo ainda é bastante residual. Deixando assim espaço aberto para novas abordagens. É o caso, nesta dissertação: a criação de um sistema de votação eletrónica capaz de responder às necessidades de uma eleição, com uma abordagem diferente dos restantes. Para tal foi usada a Framework Hyperledger Fabric. Esta facilitou bastante a criação da rede de Blockchain, assente numa estrutura coesa e fiável da Framework. Foi definido o desenho arquitetural do sistema. Este teve por base as freguesias de Portugal e teve como objetivo promover eleições a nível nacional. Uma API foi desenvolvida para dar suporte à rede eleitoral. Esta permite aos votantes exercerem o seu direito de voto através de uma aplicação que pode correr nos sues telefones. No decorrer do desenvolvimento desta dissertação foi desenvolvido um sistema de autenticação para os votantes com base no tradicional, nome de utilizador e senha. Este é um sistema de autenticação que pode trazer alguns problemas. Desde logo ser necessário criar e distribuir credenciais aos utilizadores. E confiar a qualidade das credenciais aos mesmos, podendo estas ser descobertas com mais facilidade e comprometer a qualidade do seu voto. Pensando nisto e como forma de resolver o problema, num eventual trabalho futuro, propõe-se o uso da Chave Móvel Digital[1] como mecanismo de autenticação. Seguindo esta ideia, a autenticação poderá ser realizada usando a Chave Móvel Digital ou usando o cartão de cidadão através de um leitor de cartões. Tanto a Chave Móvel Digital (CMD) e o cartão de cidadão possuem um meio de autenticação e assinatura digital certificado pelo Estado português. Permitem ao utilizador aceder a vários portais públicos ou privados, e assinar documentos digitais, com um único login.

[1]Chave Móvel Digital: https://pki.cartaodecidadao.pt/publico/politicas/PJ.CMDA_33_signed.pdf

No caso da Chave Móvel Digital, esta associa um número de telemóvel ao número de identificação civil para um cidadão português, e o número de passaporte ou título/cartão de residência para um cidadão estrangeiro. Os certificados gerados tal como acontece com a autoridade certificadora (CA) do Hyperledger Fabric utilizam o standard X.509 PKI. Torna-se assim possível utilizá-los para autenticar e identificar o votante durante o processo eleitoral. Qualquer implementação da Chave Móvel Digital necessita de aprovação e validação do estado português. Com este sistema de autenticação é possível garantir uma maior segurança em todo o processo eleitoral.

Bibliografia

[Abu+19] Yousif Abuidris et al. «Risks and Opportunities of Blockchain Based on E-Voting Systems». Em: *2019 16th International Computer Conference on Wavelet Active Media Technology and Information Processing*. 2019 16th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). Chengdu, China: IEEE, dez. de 2019, pp. 365–368. ISBN: 978-1-72814-241-8 978-1-72814-242-5. DOI: 10.1109/ICCWAMTIP47768.2019.9067529. URL: https://ieeexplore.ieee.org/document/9067529/ (acedido em 13/12/2020) (citado na página 7).

[BCD19] Emanuele Bellini, Paolo Ceravolo e Ernesto Damiani. «Blockchain-Based E-Vote-as-a-Service». Em: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). Milan, Italy: IEEE, jul. de 2019, pp. 484–486. ISBN: 978-1-72812-705-7. DOI: 10.1109/CLOUD.2019.00085. URL: https://ieeexplore.ieee.org/document/8814575/ (acedido em 31/01/2021) (citado na página 9).

[Gao+19] Shiyao Gao et al. «An Anti-Quantum E-Voting Protocol in Blockchain With Audit Function». Em: *IEEE Access* 7 (2019), pp. 115304–115316. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2935895. URL: https://ieeexplore.ieee.org/document/8804187/ (acedido em 20/12/2020) (citado na página 9).

[Hja+18] Friorik P. Hjalmarsson et al. «Blockchain-Based E-Voting System». Em: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). San Francisco, CA, USA: IEEE, jul. de 2018, pp. 983–986. ISBN: 978-1-5386-7235-8. DOI: 10.1109/CLOUD.2018.00151. URL: https://ieeexplore.ieee.org/document/8457919/ (acedido em 13/12/2020) (citado na página 9).

[KV18] Nir Kshetri e Jeffrey Voas. «Blockchain-Enabled E-Voting». Em: *IEEE Software* 35.4 (jul. de 2018), pp. 95–99. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2018.2801546. URL: https://ieeexplore.ieee.org/document/8405627/ (acedido em 13/12/2020) (citado nas páginas 7, 9).

[PBC19] Archit Pandey, Mohit Bhasi e K. Chandrasekaran. «VoteChain: A Blockchain Based E-Voting System». Em: *2019 Global Conference for Advancement in Technology (GCAT)*. 2019 Global Conference for Advancement in Technology (GCAT). BANGALURU, India: IEEE, out. de 2019, pp. 1–4. ISBN: 978-1-72813-694-3. DOI: 10.1109/GCAT47503.2019.8978295. URL: https://ieeexplore.ieee.org/document/8978295/ (acedido em 13/12/2020) (citado nas páginas 7, 9).

[PJ19] Kriti Patidar e Swapnil Jain. «Decentralized E-Voting Portal Using Blockchain». Em: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT). Kanpur, India: IEEE, jul. de 2019, pp. 1–4. ISBN: 978-1-5386-5906-9. DOI: 10.1109/ICCCNT45670.2019.8944820. URL: https://ieeexplore.ieee.org/document/8944820/ (acedido em 13/12/2020) (citado nas páginas 7, 9).

[Thu+19] Linh Vo-Cao- Thuy et al. «Votereum: An Ethereum-Based E-Voting System». Em: *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*. 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF). Danang, Vietnam: IEEE, mar. de 2019, pp. 1–6. ISBN: 978-1-5386-9313-1. DOI: 10.1109/RIVF.2019.8713661. URL: https://ieeexplore.ieee.org/document/8713661/ (acedido em 13/12/2020) (citado nas páginas 7, 9).

Apêndices

Apêndice I

Apendice - Hyperledger Fabric Scripts

```
---
OrdererOrgs:
  - Name: cne.com
    Domain: cne.com
    Specs:
      - Hostname: orderer1
        SANS:
          - 192.168.1.77

PeerOrgs:
  - Name: gsl.cne.com
    Domain: gsl.cne.com
    EnableNodeOUs: true
    CA:
      Hostname: ca1
      CommonName: ca1.gsl.cne.com
      Country: US
      Province: North Carolina
      Locality: Raleigh
    Specs:
      - Hostname: peer1
        CommonName: peer1.gsl.cne.com
        SANS:
          - 192.168.1.77
  - Name: orq.cne.com
    Domain: orq.cne.com
    EnableNodeOUs: true
    CA:
      Hostname: ca1
      CommonName: ca1.orq.cne.com
      Country: US
```

```
32        Province: North Carolina
33        Locality: Raleigh
34      Specs:
35        - Hostname: peer1
36          CommonName: peer1.orq.cne.com
37          SANS:
38            - 192.168.1.77
39    - Name: pc.cne.com
40      Domain: pc.cne.com
41      EnableNodeOUs: true
42      CA:
43        Hostname: ca1
44        CommonName: ca1.pc.cne.com
45        Country: US
46        Province: North Carolina
47        Locality: Raleigh
48      Specs:
49        - Hostname: peer1
50          CommonName: peer1.pc.cne.com
51          SANS:
52            - 192.168.1.77
53    - Name: sds.cne.com
54      Domain: sds.cne.com
55      EnableNodeOUs: true
56      CA:
57        Hostname: ca1
58        CommonName: ca1.sds.cne.com
59        Country: US
60        Province: North Carolina
61        Locality: Raleigh
62      Specs:
63        - Hostname: peer1
64          CommonName: peer1.sds.cne.com
65          SANS:
66            - 192.168.1.77
```

Listagem I.1: Configuração para geração de material criptográfico

```
1 #!/bin/bash
2 cd $FABRIC_CFG_PATH
3 # cryptogen generate --config crypto-config.yaml --output keyfiles
4 configtxgen -profile OrdererGenesis -outputBlock genesis.block -
      channelID systemchannel
5
6 configtxgen -printOrg gsl-cne-com > JoinRequest_gsl-cne-com.json
7 configtxgen -printOrg orq-cne-com > JoinRequest_orq-cne-com.json
8 configtxgen -printOrg pc-cne-com > JoinRequest_pc-cne-com.json
9 configtxgen -printOrg sds-cne-com > JoinRequest_sds-cne-com.json
```

Listagem I.2: Script de geração dos ficheiros usados para adesão das organizações

```
{
 "groups": {},
 "mod_policy": "Admins",
 "policies": {
  "Admins": {
   "mod_policy": "Admins",
   "policy": {
    "type": 1,
    "value": {
     "identities": [
      {
       "principal": {
        "msp_identifier": "orq-cne-com",
        "role": "ADMIN"
       },
       "principal_classification": "ROLE"
      }
     ],
     "rule": {
      "n_out_of": {
       "n": 1,
       "rules": [
        {
         "signed_by": 0
        }
       ]
      }
     },
     "version": 0
    }
   },
   "version": "0"
  },
  "Endorsement": {
   "mod_policy": "Admins",
   "policy": {
    "type": 1,
    "value": {
     "identities": [
      {
       "principal": {
        "msp_identifier": "orq-cne-com",
        "role": "PEER"
       },
       "principal_classification": "ROLE"
```

```
46        }
47       ],
48       "rule": {
49        "n_out_of": {
50         "n": 1,
51         "rules": [
52          {
53           "signed_by": 0
54          }
55         ]
56        }
57       },
58       "version": 0
59      }
60     },
61     "version": "0"
62    },
63    "Readers": {
64     "mod_policy": "Admins",
65     "policy": {
66      "type": 1,
67      "value": {
68       "identities": [
69        {
70         "principal": {
71          "msp_identifier": "orq-cne-com",
72          "role": "ADMIN"
73         },
74         "principal_classification": "ROLE"
75        },
76        {
77         "principal": {
78          "msp_identifier": "orq-cne-com",
79          "role": "PEER"
80         },
81         "principal_classification": "ROLE"
82        },
83        {
84         "principal": {
85          "msp_identifier": "orq-cne-com",
86          "role": "CLIENT"
87         },
88         "principal_classification": "ROLE"
89        }
90       ],
91       "rule": {
92        "n_out_of": {
93         "n": 1,
94         "rules": [
```

```
95          {
96           "signed_by": 0
97          },
98          {
99           "signed_by": 1
100         },
101         {
102          "signed_by": 2
103         }
104        ]
105       }
106      },
107      "version": 0
108     }
109    },
110    "version": "0"
111   },
112   "Writers": {
113    "mod_policy": "Admins",
114    "policy": {
115     "type": 1,
116     "value": {
117      "identities": [
118       {
119        "principal": {
120         "msp_identifier": "orq-cne-com",
121         "role": "ADMIN"
122        },
123        "principal_classification": "ROLE"
124       },
125       {
126        "principal": {
127         "msp_identifier": "orq-cne-com",
128         "role": "CLIENT"
129        },
130        "principal_classification": "ROLE"
131       }
132      ],
133      "rule": {
134       "n_out_of": {
135        "n": 1,
136        "rules": [
137         {
138          "signed_by": 0
139         },
140         {
141          "signed_by": 1
142         }
143        ]
```

```
144         }
145        },
146        "version": 0
147       }
148      },
149      "version": "0"
150     }
151    },
152    "values": {
153     "MSP": {
154      "mod_policy": "Admins",
155      "value": {
156       "config": {
157        "admins": [
158         "«CERT MUST BE HERE»"
159        ],
160        "crypto_config": {
161         "identity_identifier_hash_function": "SHA256",
162         "signature_hash_family": "SHA2"
163        },
164        "fabric_node_ous": {
165         "admin_ou_identifier": {
166          "certificate": "«CERT MUST BE HERE»",
167          "organizational_unit_identifier": "admin"
168         },
169         "client_ou_identifier": {
170          "certificate": "«CERT MUST BE HERE»",
171          "organizational_unit_identifier": "client"
172         },
173         "enable": true,
174         "orderer_ou_identifier": {
175          "certificate": "«CERT MUST BE HERE»",
176          "organizational_unit_identifier": "orderer"
177         },
178         "peer_ou_identifier": {
179          "certificate": "«CERT MUST BE HERE»",
180          "organizational_unit_identifier": "peer"
181         }
182        },
183        "intermediate_certs": [],
184        "name": "orq-cne-com",
185        "organizational_unit_identifiers": [],
186        "revocation_list": [],
187        "root_certs": [
188         "«CERT MUST BE HERE»"
189        ],
190        "signing_identity": null,
191        "tls_intermediate_certs": [],
192        "tls_root_certs": [
```

```
193         "«CERT MUST BE HERE»"
194       ]
195     },
196     "type": 0
197   },
198   "version": "0"
199  }
200 },
201 "version": "0"
202 }
```

Listagem I.3: Ficheiro usado para adesão da organização orq

```
1  #!/bin/bash
2  # Script to create channel block 0 and then create channel
3  cp $FABRIC_CFG_PATH/core.yaml /vars/core.yaml
4  cd /vars
5  export FABRIC_CFG_PATH=/vars
6  configtxgen -profile OrgChannel \
7    -outputCreateChannelTx mychannel.tx -channelID mychannel
8
9  export CORE_PEER_TLS_ENABLED=true
10 export CORE_PEER_ID=cli
11 export CORE_PEER_ADDRESS=192.168.1.77:7782
12 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
13 export CORE_PEER_LOCALMSPID=orq-cne-com
14 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
       cne.com/users/Admin@orq.cne.com/msp
15 export ORDERER_ADDRESS=192.168.1.77:7790
16 export ORDERER_TLS_CA=/vars/keyfiles/ordererOrganizations/cne.com/
       orderers/orderer1.cne.com/tls/ca.crt
17 peer channel create -c mychannel -f mychannel.tx -o $ORDERER_ADDRESS \
18   --cafile $ORDERER_TLS_CA --tls
```

Listagem I.4: Script de geração para criação do canal

```
1  # Copyright IBM Corp. All Rights Reserved.
2  #
3  # SPDX-License-Identifier: Apache-2.0
4  #
5
6  ##############################################################
7  #
8  #     Peer section
9  #
10 ##############################################################
11 peer:
12
```

```
13      # The peer id provides a name for this peer instance and is used
        when
14      # naming docker resources.
15      id: jdoe
16
17      # The networkId allows for logical separation of networks and is
        used when
18      # naming docker resources.
19      networkId: dev
20
21      # The Address at local network interface this Peer will listen on.
22      # By default, it will listen on all network interfaces
23      listenAddress: 0.0.0.0:7051
24
25      # The endpoint this peer uses to listen for inbound chaincode
        connections.
26      # If this is commented-out, the listen address is selected to be
27      # the peer's address (see below) with port 7052
28      # chaincodeListenAddress: 0.0.0.0:7052
29
30      # The endpoint the chaincode for this peer uses to connect to the
        peer.
31      # If this is not specified, the chaincodeListenAddress address is
        selected.
32      # And if chaincodeListenAddress is not specified, address is
        selected from
33      # peer address (see below). If specified peer address is invalid
        then it
34      # will fallback to the auto detected IP (local IP) regardless of
        the peer
35      # addressAutoDetect value.
36      # chaincodeAddress: 0.0.0.0:7052
37
38      # When used as peer config, this represents the endpoint to other
        peers
39      # in the same organization. For peers in other organization, see
40      # gossip.externalEndpoint for more info.
41      # When used as CLI config, this means the peer's endpoint to
        interact with
42      address: 0.0.0.0:7051
43
44      # Whether the Peer should programmatically determine its address
45      # This case is useful for docker containers.
46      # When set to true, will override peer address.
47      addressAutoDetect: false
48
49      # Keepalive settings for peer server and clients
50      keepalive:
51          # Interval is the duration after which if the server does not
```

```
    see
52          # any activity from the client it pings the client to see if
    it's alive
53          interval: 7200s
54          # Timeout is the duration the server waits for a response
55          # from the client after sending a ping before closing the
    connection
56          timeout: 20s
57          # MinInterval is the minimum permitted time between client
    pings.
58          # If clients send pings more frequently, the peer server will
59          # disconnect them
60          minInterval: 60s
61          # Client keepalive settings for communicating with other peer
    nodes
62          client:
63              # Interval is the time between pings to peer nodes.  This
    must
64              # greater than or equal to the minInterval specified by
    peer
65              # nodes
66              interval: 60s
67              # Timeout is the duration the client waits for a response
    from
68              # peer nodes before closing the connection
69              timeout: 20s
70          # DeliveryClient keepalive settings for communication with
    ordering
71          # nodes.
72          deliveryClient:
73              # Interval is the time between pings to ordering nodes.
    This must
74              # greater than or equal to the minInterval specified by
    ordering
75              # nodes.
76              interval: 60s
77              # Timeout is the duration the client waits for a response
    from
78              # ordering nodes before closing the connection
79              timeout: 20s
80
81
82      # Gossip related configuration
83      gossip:
84          # Bootstrap set to initialize gossip with.
85          # This is a list of other peers that this peer reaches out to
    at startup.
86          # Important: The endpoints here have to be endpoints of peers
    in the same
```

```
87          # organization, because the peer would refuse connecting to
    these endpoints
88          # unless they are in the same organization as the peer.
89          bootstrap: 127.0.0.1:7051
90
91          # NOTE: orgLeader and useLeaderElection parameters are mutual
    exclusive.
92          # Setting both to true would result in the termination of the
    peer
93          # since this is undefined state. If the peers are configured
    with
94          # useLeaderElection=false, make sure there is at least 1 peer
    in the
95          # organization that its orgLeader is set to true.
96
97          # Defines whenever peer will initialize dynamic algorithm for
98          # "leader" selection, where leader is the peer to establish
99          # connection with ordering service and use delivery protocol
100         # to pull ledger blocks from ordering service.
101         useLeaderElection: false
102         # Statically defines peer to be an organization "leader",
103         # where this means that current peer will maintain connection
104         # with ordering service and disseminate block across peers in
105         # its own organization. Multiple peers or all peers in an
    organization
106         # may be configured as org leaders, so that they all pull
107         # blocks directly from ordering service.
108         orgLeader: true
109
110         # Interval for membershipTracker polling
111         membershipTrackerInterval: 5s
112
113         # Overrides the endpoint that the peer publishes to peers
114         # in its organization. For peers in foreign organizations
115         # see 'externalEndpoint'
116         endpoint:
117         # Maximum count of blocks stored in memory
118         maxBlockCountToStore: 10
119         # Max time between consecutive message pushes(unit:
    millisecond)
120         maxPropagationBurstLatency: 10ms
121         # Max number of messages stored until a push is triggered to
    remote peers
122         maxPropagationBurstSize: 10
123         # Number of times a message is pushed to remote peers
124         propagateIterations: 1
125         # Number of peers selected to push messages to
126         propagatePeerNum: 3
127         # Determines frequency of pull phases(unit: second)
```

```
128         # Must be greater than digestWaitTime + responseWaitTime
129         pullInterval: 4s
130         # Number of peers to pull from
131         pullPeerNum: 3
132         # Determines frequency of pulling state info messages from
    peers(unit: second)
133         requestStateInfoInterval: 4s
134         # Determines frequency of pushing state info messages to peers
    (unit: second)
135         publishStateInfoInterval: 4s
136         # Maximum time a stateInfo message is kept until expired
137         stateInfoRetentionInterval:
138         # Time from startup certificates are included in Alive
    messages(unit: second)
139         publishCertPeriod: 10s
140         # Should we skip verifying block messages or not (currently
    not in use)
141         skipBlockVerification: false
142         # Dial timeout(unit: second)
143         dialTimeout: 3s
144         # Connection timeout(unit: second)
145         connTimeout: 2s
146         # Buffer size of received messages
147         recvBuffSize: 20
148         # Buffer size of sending messages
149         sendBuffSize: 200
150         # Time to wait before pull engine processes incoming digests (
    unit: second)
151         # Should be slightly smaller than requestWaitTime
152         digestWaitTime: 1s
153         # Time to wait before pull engine removes incoming nonce (unit
    : milliseconds)
154         # Should be slightly bigger than digestWaitTime
155         requestWaitTime: 1500ms
156         # Time to wait before pull engine ends pull (unit: second)
157         responseWaitTime: 2s
158         # Alive check interval(unit: second)
159         aliveTimeInterval: 5s
160         # Alive expiration timeout(unit: second)
161         aliveExpirationTimeout: 25s
162         # Reconnect interval(unit: second)
163         reconnectInterval: 25s
164         # Max number of attempts to connect to a peer
165         maxConnectionAttempts: 120
166         # Message expiration factor for alive messages
167         msgExpirationFactor: 20
168         # This is an endpoint that is published to peers outside of
    the organization.
169         # If this isn't set, the peer will not be known to other
```

```
    organizations.
170         externalEndpoint:
171         # Leader election service configuration
172         election:
173             # Longest time peer waits for stable membership during
    leader election startup (unit: second)
174             startupGracePeriod: 15s
175             # Interval gossip membership samples to check its
    stability (unit: second)
176             membershipSampleInterval: 1s
177             # Time passes since last declaration message before peer
    decides to perform leader election (unit: second)
178             leaderAliveThreshold: 10s
179             # Time between peer sends propose message and declares
    itself as a leader (sends declaration message) (unit: second)
180             leaderElectionDuration: 5s
181
182         pvtData:
183             # pullRetryThreshold determines the maximum duration of
    time private data corresponding for a given block
184             # would be attempted to be pulled from peers until the
    block would be committed without the private data
185             pullRetryThreshold: 60s
186             # As private data enters the transient store, it is
    associated with the peer's ledger's height at that time.
187             # transientstoreMaxBlockRetention defines the maximum
    difference between the current ledger's height upon commit,
188             # and the private data residing inside the transient store
     that is guaranteed not to be purged.
189             # Private data is purged from the transient store when
    blocks with sequences that are multiples
190             # of transientstoreMaxBlockRetention are committed.
191             transientstoreMaxBlockRetention: 1000
192             # pushAckTimeout is the maximum time to wait for an
    acknowledgement from each peer
193             # at private data push at endorsement time.
194             pushAckTimeout: 3s
195             # Block to live pulling margin, used as a buffer
196             # to prevent peer from trying to pull private data
197             # from peers that is soon to be purged in next N blocks.
198             # This helps a newly joined peer catch up to current
199             # blockchain height quicker.
200             btlPullMargin: 10
201             # the process of reconciliation is done in an endless loop
    , while in each iteration reconciler tries to
202             # pull from the other peers the most recent missing blocks
     with a maximum batch size limitation.
203             # reconcileBatchSize determines the maximum batch size of
    missing private data that will be reconciled in a
```

```
204             # single iteration.
205             reconcileBatchSize: 10
206             # reconcileSleepInterval determines the time reconciler
    sleeps from end of an iteration until the beginning
207             # of the next reconciliation iteration.
208             reconcileSleepInterval: 1m
209             # reconciliationEnabled is a flag that indicates whether
    private data reconciliation is enable or not.
210             reconciliationEnabled: true
211             # skipPullingInvalidTransactionsDuringCommit is a flag
    that indicates whether pulling of invalid
212             # transaction's private data from other peers need to be
    skipped during the commit time and pulled
213             # only through reconciler.
214             skipPullingInvalidTransactionsDuringCommit: false
215             # implicitCollectionDisseminationPolicy specifies the
    dissemination  policy for the peer's own implicit collection.
216             # When a peer endorses a proposal that writes to its own
    implicit collection, below values override the default values
217             # for disseminating private data.
218             # Note that it is applicable to all channels the peer has
    joined. The implication is that requiredPeerCount has to
219             # be smaller than the number of peers in a channel that
    has the lowest numbers of peers from the organization.
220             implicitCollectionDisseminationPolicy:
221                # requiredPeerCount defines the minimum number of
    eligible peers to which the peer must successfully
222                # disseminate private data for its own implicit
    collection during endorsement. Default value is 0.
223                requiredPeerCount: 0
224                # maxPeerCount defines the maximum number of eligible
    peers to which the peer will attempt to
225                # disseminate private data for its own implicit
    collection during endorsement. Default value is 1.
226                maxPeerCount: 1
227
228         # Gossip state transfer related configuration
229         state:
230             # indicates whenever state transfer is enabled or not
231             # default value is true, i.e. state transfer is active
232             # and takes care to sync up missing blocks allowing
233             # lagging peer to catch up to speed with rest network
234             enabled: false
235             # checkInterval interval to check whether peer is lagging
    behind enough to
236             # request blocks via state transfer from another peer.
237             checkInterval: 10s
238             # responseTimeout amount of time to wait for state
    transfer response from
```

```
239             # other peers
240             responseTimeout: 3s
241             # batchSize the number of blocks to request via state
    transfer from another peer
242             batchSize: 10
243             # blockBufferSize reflects the size of the re-ordering
    buffer
244             # which captures blocks and takes care to deliver them in
    order
245             # down to the ledger layer. The actual buffer size is
    bounded between
246             # 0 and 2*blockBufferSize, each channel maintains its own
    buffer
247             blockBufferSize: 20
248             # maxRetries maximum number of re-tries to ask
249             # for single state transfer request
250             maxRetries: 3
251
252     # TLS Settings
253     tls:
254         # Require server-side TLS
255         enabled:  false
256         # Require client certificates / mutual TLS for inbound
    connections.
257         # Note that clients that are not configured to use a
    certificate will
258         # fail to connect to the peer.
259         clientAuthRequired: false
260         # X.509 certificate used for TLS server
261         cert:
262             file: tls/server.crt
263         # Private key used for TLS server
264         key:
265             file: tls/server.key
266         # rootcert.file represents the trusted root certificate chain
    used for verifying certificates
267         # of other nodes during outbound connections.
268         # It is not required to be set, but can be used to augment the
     set of TLS CA certificates
269         # available from the MSPs of each channels configuration.
270         rootcert:
271             file: tls/ca.crt
272         # If mutual TLS is enabled, clientRootCAs.files contains a
    list of additional root certificates
273         # used for verifying certificates of client connections.
274         # It augments the set of TLS CA certificates available from
    the MSPs of each channels configuration.
275         # Minimally, set your organization's TLS CA root certificate
    so that the peer can receive join channel requests.
```

```
276         clientRootCAs:
277             files:
278               - tls/ca.crt
279         # Private key used for TLS when making client connections.
280         # If not set, peer.tls.key.file will be used instead
281         clientKey:
282             file:
283         # X.509 certificate used for TLS when making client
    connections.
284         # If not set, peer.tls.cert.file will be used instead
285         clientCert:
286             file:
287
288     # Authentication contains configuration parameters related to
    authenticating
289     # client messages
290     authentication:
291         # the acceptable difference between the current server time
    and the
292         # client's time as specified in a client request message
293         timewindow: 15m
294
295     # Path on the file system where peer will store data (eg ledger).
    This
296     # location must be access control protected to prevent unintended
297     # modification that might corrupt the peer operations.
298     fileSystemPath: /var/hyperledger/production
299
300     # BCCSP (Blockchain crypto provider): Select which crypto
    implementation or
301     # library to use
302     BCCSP:
303         Default: SW
304         # Settings for the SW crypto provider (i.e. when DEFAULT: SW)
305         SW:
306             # TODO: The default Hash and Security level needs
    refactoring to be
307             # fully configurable. Changing these defaults requires
    coordination
308             # SHA2 is hardcoded in several places, not only BCCSP
309             Hash: SHA2
310             Security: 256
311             # Location of Key Store
312             FileKeyStore:
313                 # If "", defaults to 'mspConfigPath'/keystore
314                 KeyStore:
315         # Settings for the PKCS#11 crypto provider (i.e. when DEFAULT:
     PKCS11)
316         PKCS11:
```

```
317             # Location of the PKCS11 module library
318             Library:
319             # Token Label
320             Label:
321             # User PIN
322             Pin:
323             Hash:
324             Security:
325
326     # Path on the file system where peer will find MSP local
        configurations
327     mspConfigPath: msp
328
329     # Identifier of the local MSP
330     # ----!!!!IMPORTANT!!! -!!!IMPORTANT!!! -!!!IMPORTANT!!!!----
331     # Deployers need to change the value of the localMspId string.
332     # In particular, the name of the local MSP ID of a peer needs
333     # to match the name of one of the MSPs in each of the channel
334     # that this peer is a member of. Otherwise this peer's messages
335     # will not be identified as valid by other nodes.
336     localMspId: SampleOrg
337
338     # CLI common client config options
339     client:
340         # connection timeout
341         connTimeout: 3s
342
343     # Delivery service related config
344     deliveryclient:
345         # It sets the total time the delivery service may spend in
        reconnection
346         # attempts until its retry logic gives up and returns an error
347         reconnectTotalTimeThreshold: 3600s
348
349         # It sets the delivery service <-> ordering service node
        connection timeout
350         connTimeout: 3s
351
352         # It sets the delivery service maximal delay between
        consecutive retries
353         reConnectBackoffThreshold: 3600s
354
355         # A list of orderer endpoint addresses which should be
        overridden
356         # when found in channel configurations.
357         addressOverrides:
358         #  - from:
359         #    to:
360         #    caCertsFile:
```

```
361         #  - from:
362         #    to:
363         #    caCertsFile:
364
365     # Type for the local MSP - by default it's of type bccsp
366     localMspType: bccsp
367
368     # Used with Go profiling tools only in none production environment
        . In
369     # production, it should be disabled (eg enabled: false)
370     profile:
371         enabled:     false
372         listenAddress: 0.0.0.0:6060
373
374     # Handlers defines custom handlers that can filter and mutate
375     # objects passing within the peer, such as:
376     #   Auth filter - reject or forward proposals from clients
377     #   Decorators  - append or mutate the chaincode input passed to
        the chaincode
378     #   Endorsers   - Custom signing over proposal response payload
        and its mutation
379     # Valid handler definition contains:
380     #   - A name which is a factory method name defined in
381     #     core/handlers/library/library.go for statically compiled
        handlers
382     #   - library path to shared object binary for pluggable filters
383     # Auth filters and decorators are chained and executed in the
        order that
384     # they are defined. For example:
385     # authFilters:
386     #   -
387     #     name: FilterOne
388     #     library: /opt/lib/filter.so
389     #   -
390     #     name: FilterTwo
391     # decorators:
392     #   -
393     #     name: DecoratorOne
394     #   -
395     #     name: DecoratorTwo
396     #     library: /opt/lib/decorator.so
397     # Endorsers are configured as a map that its keys are the
        endorsement system chaincodes that are being overridden.
398     # Below is an example that overrides the default ESCC and uses an
        endorsement plugin that has the same functionality
399     # as the default ESCC.
400     # If the 'library' property is missing, the name is used as the
        constructor method in the builtin library similar
401     # to auth filters and decorators.
```

```
402     # endorsers:
403     #    escc:
404     #      name: DefaultESCC
405     #      library: /etc/hyperledger/fabric/plugin/escc.so
406     handlers:
407         authFilters:
408           -
409             name: DefaultAuth
410           -
411             name: ExpirationCheck    # This filter checks identity
        x509 certificate expiration
412         decorators:
413           -
414             name: DefaultDecorator
415         endorsers:
416           escc:
417             name: DefaultEndorsement
418             library:
419         validators:
420           vscc:
421             name: DefaultValidation
422             library:
423
424     #    library: /etc/hyperledger/fabric/plugin/escc.so
425     # Number of goroutines that will execute transaction validation in
         parallel.
426     # By default, the peer chooses the number of CPUs on the machine.
        Set this
427     # variable to override that choice.
428     # NOTE: overriding this value might negatively influence the
        performance of
429     # the peer so please change this value only if you know what you'
        re doing
430     validatorPoolSize:
431
432     # The discovery service is used by clients to query information
        about peers,
433     # such as - which peers have joined a certain channel, what is the
         latest
434     # channel config, and most importantly - given a chaincode and a
        channel,
435     # what possible sets of peers satisfy the endorsement policy.
436     discovery:
437         enabled: true
438         # Whether the authentication cache is enabled or not.
439         authCacheEnabled: true
440         # The maximum size of the cache, after which a purge takes
        place
441         authCacheMaxSize: 1000
```

```
442         # The proportion (0 to 1) of entries that remain in the cache
    after the cache is purged due to overpopulation
443         authCachePurgeRetentionRatio: 0.75
444         # Whether to allow non-admins to perform non channel scoped
    queries.
445         # When this is false, it means that only peer admins can
    perform non channel scoped queries.
446         orgMembersAllowedAccess: false
447
448     # Limits is used to configure some internal resource limits.
449     limits:
450         # Concurrency limits the number of concurrently running
    requests to a service on each peer.
451         # Currently this option is only applied to endorser service
    and deliver service.
452         # When the property is missing or the value is 0, the
    concurrency limit is disabled for the service.
453         concurrency:
454             # endorserService limits concurrent requests to endorser
    service that handles chaincode deployment, query and invocation,
455             # including both user chaincodes and system chaincodes.
456             endorserService: 2500
457             # deliverService limits concurrent event listeners
    registered to deliver service for blocks and transaction events.
458             deliverService: 2500
459
460############################################################
461#
462#    VM section
463#
464############################################################
465vm:
466
467     # Endpoint of the vm management system.  For docker can be one of
    the following in general
468     # unix:///var/run/docker.sock
469     # http://localhost:2375
470     # https://localhost:2376
471     endpoint: unix:///var/run/docker.sock
472
473     # settings for docker vms
474     docker:
475         tls:
476             enabled: false
477             ca:
478                 file: docker/ca.crt
479             cert:
480                 file: docker/tls.crt
481             key:
```

```
482                 file: docker/tls.key
483
484         # Enables/disables the standard out/err from chaincode
    containers for
485         # debugging purposes
486         attachStdout: false
487
488         # Parameters on creating docker container.
489         # Container may be efficiently created using ipam & dns-server
     for cluster
490         # NetworkMode - sets the networking mode for the container.
    Supported
491         # standard values are: 'host'(default),'bridge','ipvlan','none
    '.
492         # Dns - a list of DNS servers for the container to use.
493         # Note: 'Privileged' 'Binds' 'Links' and 'PortBindings'
    properties of
494         # Docker Host Config are not supported and will not be used if
     set.
495         # LogConfig - sets the logging driver (Type) and related
    options
496         # (Config) for Docker. For more info,
497         # https://docs.docker.com/engine/admin/logging/overview/
498         # Note: Set LogConfig using Environment Variables is not
    supported.
499         hostConfig:
500             NetworkMode: host
501             Dns:
502                #- 192.168.0.1
503             LogConfig:
504                 Type: json-file
505                 Config:
506                     max-size: "50m"
507                     max-file: "5"
508             Memory: 2147483648
509
510###############################################################
511#
512#    Chaincode section
513#
514###############################################################
515chaincode:
516
517    # The id is used by the Chaincode stub to register the executing
    Chaincode
518    # ID with the Peer and is generally supplied through ENV variables
519    # the 'path' form of ID is provided when installing the chaincode.
520    # The 'name' is used for all other requests and can be any string.
521    id:
```

```
522         path:
523         name:
524
525     # Generic builder environment, suitable for most chaincode types
526     builder: $(DOCKER_NS)/fabric-ccenv:$(TWO_DIGIT_VERSION)
527
528     # Enables/disables force pulling of the base docker images (listed
        below)
529     # during user chaincode instantiation.
530     # Useful when using moving image tags (such as :latest)
531     pull: false
532
533     golang:
534         # golang will never need more than baseos
535         runtime: $(DOCKER_NS)/fabric-baseos:$(TWO_DIGIT_VERSION)
536
537         # whether or not golang chaincode should be linked dynamically
538         dynamicLink: false
539
540     java:
541         # This is an image based on java:openjdk-8 with addition
        compiler
542         # tools added for java shim layer packaging.
543         # This image is packed with shim layer libraries that are
        necessary
544         # for Java chaincode runtime.
545         runtime: $(DOCKER_NS)/fabric-javaenv:$(TWO_DIGIT_VERSION)
546
547     node:
548         # This is an image based on node:$(NODE_VER)-alpine
549         runtime: $(DOCKER_NS)/fabric-nodeenv:$(TWO_DIGIT_VERSION)
550
551     # List of directories to treat as external builders and launchers
        for
552     # chaincode. The external builder detection processing will
        iterate over the
553     # builders in the order specified below.
554     externalBuilders: []
555         # - path: /path/to/directory
556         #   name: descriptive-builder-name
557         #   propagateEnvironment:
558         #       - ENVVAR_NAME_TO_PROPAGATE_FROM_PEER
559         #       - GOPROXY
560
561     # The maximum duration to wait for the chaincode build and install
        process
562     # to complete.
563     installTimeout: 300s
564
```

```
565     # Timeout duration for starting up a container and waiting for
        Register
566     # to come through.
567     startuptimeout: 300s
568
569     # Timeout duration for Invoke and Init calls to prevent runaway.
570     # This timeout is used by all chaincodes in all the channels,
        including
571     # system chaincodes.
572     # Note that during Invoke, if the image is not available (e.g.
        being
573     # cleaned up when in development environment), the peer will
        automatically
574     # build the image, which might take more time. In production
        environment,
575     # the chaincode image is unlikely to be deleted, so the timeout
        could be
576     # reduced accordingly.
577     executetimeout: 30s
578
579     # There are 2 modes: "dev" and "net".
580     # In dev mode, user runs the chaincode after starting peer from
581     # command line on local machine.
582     # In net mode, peer will run chaincode in a docker container.
583     mode: net
584
585     # keepalive in seconds. In situations where the communication goes
         through a
586     # proxy that does not support keep-alive, this parameter will
        maintain connection
587     # between peer and chaincode.
588     # A value <= 0 turns keepalive off
589     keepalive: 0
590
591     # enabled system chaincodes
592     system:
593         _lifecycle: enable
594         cscc: enable
595         lscc: enable
596         qscc: enable
597
598     # Logging section for the chaincode container
599     logging:
600       # Default level for all loggers within the chaincode container
601       level:  info
602       # Override default level for the 'shim' logger
603       shim:   warning
604       # Format for the chaincode container logs
605       format: '%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}]
```

```
      %{shortfunc} -> %{level:.4s} %{id:03x}%{color:reset} %{message}'
606
607 ###############################################################
608 #
609 #    Ledger section - ledger configuration encompasses both the
    blockchain
610 #    and the state
611 #
612 ###############################################################
613 ledger:
614
615   blockchain:
616
617   state:
618     # stateDatabase - options are "goleveldb", "CouchDB"
619     # goleveldb - default state database stored in goleveldb.
620     # CouchDB - store state database in CouchDB
621     stateDatabase: goleveldb
622     # Limit on the number of records to return per query
623     totalQueryLimit: 100000
624     couchDBConfig:
625        # It is recommended to run CouchDB on the same server as the
    peer, and
626        # not map the CouchDB container port to a server port in docker
    -compose.
627        # Otherwise proper security must be provided on the connection
    between
628        # CouchDB client (on the peer) and server.
629        couchDBAddress: 127.0.0.1:5984
630        # This username must have read and write authority on CouchDB
631        username:
632        # The password is recommended to pass as an environment
    variable
633        # during start up (eg CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD)
    .
634        # If it is stored here, the file must be access control
    protected
635        # to prevent unintended users from discovering the password.
636        password:
637        # Number of retries for CouchDB errors
638        maxRetries: 3
639        # Number of retries for CouchDB errors during peer startup.
640        # The delay between retries doubles for each attempt.
641        # Default of 10 retries results in 11 attempts over 2 minutes.
642        maxRetriesOnStartup: 10
643        # CouchDB request timeout (unit: duration, e.g. 20s)
644        requestTimeout: 35s
645        # Limit on the number of records per each CouchDB query
646        # Note that chaincode queries are only bound by totalQueryLimit
```

```
    .
647         # Internally the chaincode may execute multiple CouchDB queries
    ,
648         # each of size internalQueryLimit.
649         internalQueryLimit: 1000
650         # Limit on the number of records per CouchDB bulk update batch
651         maxBatchUpdateSize: 1000
652         # Warm indexes after every N blocks.
653         # This option warms any indexes that have been
654         # deployed to CouchDB after every N blocks.
655         # A value of 1 will warm indexes after every block commit,
656         # to ensure fast selector queries.
657         # Increasing the value may improve write efficiency of peer and
     CouchDB,
658         # but may degrade query response time.
659         warmIndexesAfterNBlocks: 1
660         # Create the _global_changes system database
661         # This is optional.  Creating the global changes database will
    require
662         # additional system resources to track changes and maintain the
     database
663         createGlobalChangesDB: false
664         # CacheSize denotes the maximum mega bytes (MB) to be allocated
     for the in-memory state
665         # cache. Note that CacheSize needs to be a multiple of 32 MB.
    If it is not a multiple
666         # of 32 MB, the peer would round the size to the next multiple
    of 32 MB.
667         # To disable the cache, 0 MB needs to be assigned to the
    cacheSize.
668         cacheSize: 64
669
670   history:
671     # enableHistoryDatabase - options are true or false
672     # Indicates if the history of key updates should be stored.
673     # All history 'index' will be stored in goleveldb, regardless if
    using
674     # CouchDB or alternate database for the state.
675     enableHistoryDatabase: true
676
677   pvtdataStore:
678     # the maximum db batch size for converting
679     # the ineligible missing data entries to eligible missing data
    entries
680     collElgProcMaxDbBatchSize: 5000
681     # the minimum duration (in milliseconds) between writing
682     # two consecutive db batches for converting the ineligible missing
     data entries to eligible missing data entries
683     collElgProcDbBatchesInterval: 1000
```

```
684     # The missing data entries are classified into two categories:
685     # (1) prioritized
686     # (2) deprioritized
687     # Initially, all missing data are in the prioritized list. When
        the
688     # reconciler is unable to fetch the missing data from other peers,
689     # the unreconciled missing data would be moved to the
        deprioritized list.
690     # The reconciler would retry deprioritized missing data after
        every
691     # deprioritizedDataReconcilerInterval (unit: minutes). Note that
        the
692     # interval needs to be greater than the reconcileSleepInterval
693     deprioritizedDataReconcilerInterval: 60m
694
695   snapshots:
696     # Path on the file system where peer will store ledger snapshots
697     rootDir: /var/hyperledger/production/snapshots
698
699##################################################
700#
701#    Operations section
702#
703##################################################
704operations:
705     # host and port for the operations server
706     listenAddress: 127.0.0.1:9443
707
708     # TLS configuration for the operations endpoint
709     tls:
710         # TLS enabled
711         enabled: false
712
713         # path to PEM encoded server certificate for the operations
        server
714         cert:
715             file:
716
717         # path to PEM encoded server key for the operations server
718         key:
719             file:
720
721         # most operations service endpoints require client
        authentication when TLS
722         # is enabled. clientAuthRequired requires client certificate
        authentication
723         # at the TLS layer to access all resources.
724         clientAuthRequired: false
725
```

```
726         # paths to PEM encoded ca certificates to trust for client
    authentication
727         clientRootCAs:
728             files: []
729
730 ###############################################################
731 #
732 #     Metrics section
733 #
734 ###############################################################
735 metrics:
736     # metrics provider is one of statsd, prometheus, or disabled
737     provider: disabled
738
739     # statsd configuration
740     statsd:
741         # network type: tcp or udp
742         network: udp
743
744         # statsd server address
745         address: 127.0.0.1:8125
746
747         # the interval at which locally cached counters and gauges are
     pushed
748         # to statsd; timings are pushed immediately
749         writeInterval: 10s
750
751         # prefix is prepended to all emitted statsd metrics
752         prefix:
```

Listagem I.5: Ficheiro de configuração obtido do Hyperledger Fabric

```
1  #!/bin/bash
2  # Script to join a peer to a channel
3  export CORE_PEER_TLS_ENABLED=true
4  export CORE_PEER_ID=cli
5  export CORE_PEER_ADDRESS=192.168.1.77:7782
6  export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
7  export CORE_PEER_LOCALMSPID=orq-cne-com
8  export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
       cne.com/users/Admin@orq.cne.com/msp
9  export ORDERER_ADDRESS=192.168.1.77:7790
10 export ORDERER_TLS_CA=/vars/keyfiles/ordererOrganizations/cne.com/
       orderers/orderer1.cne.com/tls/ca.crt
11 if [ ! -f "mychannel.genesis.block" ]; then
12   peer channel fetch oldest -o $ORDERER_ADDRESS --cafile
       $ORDERER_TLS_CA \
13   --tls -c mychannel /vars/mychannel.genesis.block
```

```
14 fi
15
16 peer channel join -b /vars/mychannel.genesis.block \
17   -o $ORDERER_ADDRESS --cafile $ORDERER_TLS_CA --tls
```

Listagem I.6: Script de adição dos peers da organização orq ao canal

```
1  #!/bin/bash
2  # Script to instantiate chaincode
3  cp $FABRIC_CFG_PATH/core.yaml /vars/core.yaml
4  cd /vars
5  export FABRIC_CFG_PATH=/vars
6
7  export CORE_PEER_TLS_ENABLED=true
8  export CORE_PEER_ID=cli
9  export CORE_PEER_ADDRESS=192.168.1.77:7782
10 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
11 export CORE_PEER_LOCALMSPID=orq-cne-com
12 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
       cne.com/users/Admin@orq.cne.com/msp
13 export ORDERER_ADDRESS=192.168.1.77:7790
14 export ORDERER_TLS_CA=/vars/keyfiles/ordererOrganizations/cne.com/
       orderers/orderer1.cne.com/tls/ca.crt
15
16 # 1. Fetch the channel configuration
17 peer channel fetch config config_block.pb -o $ORDERER_ADDRESS \
18   --cafile $ORDERER_TLS_CA --tls -c mychannel
19
20 # 2. Translate the configuration into json format
21 configtxlator proto_decode --input config_block.pb --type common.Block
       \
22   | jq .data.data[0].payload.data.config > mychannel_current_config.
       json
23 echo "--<<-->>--"
24
25 # 3. Update the current config in json with the organization anchor
       peer we want to add
26 jq '.channel_group.groups.Application.groups."orq-cne-com".values +=
       {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"
       host": "192.168.1.77","port": 7782}]},"version": "0"}}'
       mychannel_current_config.json > mychannel_modified_anchor_config.
       json
27
28 # 4. Translate the current config in json format to protobuf format
29 configtxlator proto_encode --input mychannel_current_config.json \
30   --type common.Config --output config.pb
31
32 # 5. Translate the desired config in json format to protobuf format
```

```
33 configtxlator proto_encode --input mychannel_modified_anchor_config.
       json \
34   --type common.Config --output modified_config.pb
35
36 # 6. Calculate the delta of the current config and desired config
37 configtxlator compute_update --channel_id mychannel \
38   --original config.pb --updated modified_config.pb \
39   --output mychannel_anchor_update.pb
40
41 # 7. Decode the delta of the config to json format
42 configtxlator proto_decode --input mychannel_anchor_update.pb \
43   --type common.ConfigUpdate | jq . > mychannel_anchor_update.json
44
45 # 8. Now wrap of the delta config to fabric envelop block
46 echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel
       ", "type":2}},"data":{"config_update":'$(cat
       mychannel_anchor_update.json)'}}}' | jq . >
       mychannel_anchor_update_envelope.json
47
48 # 9. Encode the json format into protobuf format
49 configtxlator proto_encode --input mychannel_anchor_update_envelope.
       json \
50   --type common.Envelope --output mychannel_anchor_update_envelope.pb
51
52 # 10. Need to sign anchor update envelop by org admin
53 peer channel update -o $ORDERER_ADDRESS --tls --cafile $ORDERER_TLS_CA
       \
54   -f mychannel_anchor_update_envelope.pb -c mychannel
```

Listagem I.7: Script de definição do peer ancora da organização.

```
1 #!/bin/bash
2 # Script to install chaincode onto a peer node
3 export CORE_PEER_TLS_ENABLED=true
4 export CORE_PEER_ID=cli
5 export CORE_PEER_ADDRESS=192.168.1.77:7782
6 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
      orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
7 export CORE_PEER_LOCALMSPID=orq-cne-com
8 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
      cne.com/users/Admin@orq.cne.com/msp
9 cd /go/src/github.com/chaincode/fabcar
10
11
12 if [ ! -f "fabcar_node_1.0.tar.gz" ]; then
13   peer lifecycle chaincode package fabcar_node_1.0.tar.gz \
14     -p /go/src/github.com/chaincode/fabcar/node/ \
15     --lang node --label fabcar_1.0
16 fi
```

```
17
18 peer lifecycle chaincode install fabcar_node_1.0.tar.gz
```

Listagem I.8: Script de instalção do chain code nos peers da organização ńorqż

```
1  #!/bin/bash
2  # Script to approve chaincode
3  export CORE_PEER_TLS_ENABLED=true
4  export CORE_PEER_ID=cli
5  export CORE_PEER_ADDRESS=192.168.1.77:7782
6  export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
7  export CORE_PEER_LOCALMSPID=orq-cne-com
8  export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
       cne.com/users/Admin@orq.cne.com/msp
9  export ORDERER_ADDRESS=192.168.1.77:7790
10 export ORDERER_TLS_CA=/vars/keyfiles/ordererOrganizations/cne.com/
       orderers/orderer1.cne.com/tls/ca.crt
11
12 peer lifecycle chaincode queryinstalled -O json | jq -r '.
       installed_chaincodes | .[] | select(.package_id|startswith("
       fabcar_1.0:"))' > ccstatus.json
13
14 PKID=$(jq '.package_id' ccstatus.json | xargs)
15 REF=$(jq '.references.mychannel' ccstatus.json)
16
17 SID=$(peer lifecycle chaincode querycommitted -C mychannel -O json \
18   | jq -r '.chaincode_definitions|.[]|select(.name=="fabcar")|.
       sequence' || true)
19 if [[ -z $SID ]]; then
20   SEQUENCE=1
21 elif [[ -z $REF ]]; then
22   SEQUENCE=$SID
23 else
24   SEQUENCE=$((1+$SID))
25 fi
26
27
28 export CORE_PEER_LOCALMSPID=gsl-cne-com
29 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       gsl.cne.com/peers/peer1.gsl.cne.com/tls/ca.crt
30 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/gsl.
       cne.com/users/Admin@gsl.cne.com/msp
31 export CORE_PEER_ADDRESS=192.168.1.77:7785
32
33 # approved=$(peer lifecycle chaincode checkcommitreadiness --channelID
       mychannel \
34 #   --name fabcar --version 1.0 --init-required --sequence $SEQUENCE
       --tls \
```

```
35 #   --cafile $ORDERER_TLS_CA --output json | jq -r '.approvals.gsl-cne
       -com')
36
37 # if [[ "$approved" == "false" ]]; then
38   peer lifecycle chaincode approveformyorg --channelID mychannel --
       name fabcar \
39     --version 1.0 --package-id $PKID \
40   --init-required \
41     --sequence $SEQUENCE -o $ORDERER_ADDRESS --tls --cafile
       $ORDERER_TLS_CA
42 # fi
43
44 export CORE_PEER_LOCALMSPID=orq-cne-com
45 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/
       orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
46 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.
       cne.com/users/Admin@orq.cne.com/msp
47 export CORE_PEER_ADDRESS=192.168.1.77:7782
48
49 # approved=$(peer lifecycle chaincode checkcommitreadiness --channelID
        mychannel \
50 #   --name fabcar --version 1.0 --init-required --sequence $SEQUENCE
       --tls \
51 #   --cafile $ORDERER_TLS_CA --output json | jq -r '.approvals.orq-cne
       -com')
52
53 # if [[ "$approved" == "false" ]]; then
54   peer lifecycle chaincode approveformyorg --channelID mychannel --
       name fabcar \
55     --version 1.0 --package-id $PKID \
56   --init-required \
57     --sequence $SEQUENCE -o $ORDERER_ADDRESS --tls --cafile
       $ORDERER_TLS_CA
58 # fi
59
60 export CORE_PEER_LOCALMSPID=pc-cne-com
61 export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/pc
       .cne.com/peers/peer1.pc.cne.com/tls/ca.crt
62 export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/pc.cne
       .com/users/Admin@pc.cne.com/msp
63 export CORE_PEER_ADDRESS=192.168.1.77:7784
64
65 # approved=$(peer lifecycle chaincode checkcommitreadiness --channelID
        mychannel \
66 #   --name fabcar --version 1.0 --init-required --sequence $SEQUENCE
       --tls \
67 #   --cafile $ORDERER_TLS_CA --output json | jq -r '.approvals.pc-cne-
       com')
68
```

```
# if [[ "$approved" == "false" ]]; then
  peer lifecycle chaincode approveformyorg --channelID mychannel --name fabcar \
    --version 1.0 --package-id $PKID \
  --init-required \
    --sequence $SEQUENCE -o $ORDERER_ADDRESS --tls --cafile $ORDERER_TLS_CA
# fi

export CORE_PEER_LOCALMSPID=sds-cne-com
export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/sds.cne.com/peers/peer1.sds.cne.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/sds.cne.com/users/Admin@sds.cne.com/msp
export CORE_PEER_ADDRESS=192.168.1.77:7783

# approved=$(peer lifecycle chaincode checkcommitreadiness --channelID mychannel \
#   --name fabcar --version 1.0 --init-required --sequence $SEQUENCE --tls \
#   --cafile $ORDERER_TLS_CA --output json | jq -r '.approvals.sds-cne-com')

# if [[ "$approved" == "false" ]]; then
  peer lifecycle chaincode approveformyorg --channelID mychannel --name fabcar \
    --version 1.0 --package-id $PKID \
  --init-required \
    --sequence $SEQUENCE -o $ORDERER_ADDRESS --tls --cafile $ORDERER_TLS_CA
# fi
```

Listagem I.9: Script de aprovação do chaincode

```
#!/bin/bash
# Script to instantiate chaincode
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_ID=cli
export CORE_PEER_ADDRESS=192.168.1.77:7782
export CORE_PEER_TLS_ROOTCERT_FILE=/vars/keyfiles/peerOrganizations/orq.cne.com/peers/peer1.orq.cne.com/tls/ca.crt
export CORE_PEER_LOCALMSPID=orq-cne-com
export CORE_PEER_MSPCONFIGPATH=/vars/keyfiles/peerOrganizations/orq.cne.com/users/Admin@orq.cne.com/msp
export ORDERER_ADDRESS=192.168.1.77:7790
export ORDERER_TLS_CA=/vars/keyfiles/ordererOrganizations/cne.com/orderers/orderer1.cne.com/tls/ca.crt
SID=$(peer lifecycle chaincode querycommitted -C mychannel -O json \
```

```
12    | jq -r '.chaincode_definitions|.[]|select(.name=="fabcar")|.
      sequence' || true)
13
14 if [[ -z $SID ]]; then
15   SEQUENCE=1
16 else
17   SEQUENCE=$((1+$SID))
18 fi
19
20 peer lifecycle chaincode commit -o $ORDERER_ADDRESS --channelID
      mychannel \
21   --name fabcar --version 1.0 --sequence $SEQUENCE \
22   --peerAddresses 192.168.1.77:7785 \
23   --tlsRootCertFiles /vars/keyfiles/peerOrganizations/gsl.cne.com/
      peers/peer1.gsl.cne.com/tls/ca.crt \
24   --peerAddresses 192.168.1.77:7782 \
25   --tlsRootCertFiles /vars/keyfiles/peerOrganizations/orq.cne.com/
      peers/peer1.orq.cne.com/tls/ca.crt \
26   --peerAddresses 192.168.1.77:7784 \
27   --tlsRootCertFiles /vars/keyfiles/peerOrganizations/pc.cne.com/peers
      /peer1.pc.cne.com/tls/ca.crt \
28   --peerAddresses 192.168.1.77:7783 \
29   --tlsRootCertFiles /vars/keyfiles/peerOrganizations/sds.cne.com/
      peers/peer1.sds.cne.com/tls/ca.crt \
30   --init-required \
31   --cafile $ORDERER_TLS_CA --tls
```

Listagem I.10: Script de envio da definição do chaincode ao canal

Apêndice II

Apendice - SDK APIs

```
'use strict';

const axios = require('axios');
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const morgan = require('morgan');
const util = require('util');
const path = require('path');
const fs = require('fs');

let network = require('./fabric/network.js');
const { token } = require('morgan');

const app = express();
app.use(morgan('combined'));
app.use(bodyParser.json());

var corsOptions = {
    origin: '*',
    optionsSuccessStatus: 200 // some legacy browsers (IE11, various
    SmartTVs) choke on 204
}


app.use(cors(corsOptions));

const configPath = path.join(process.cwd(), '../profiles/
    mychannel_connection_for_nodesdk.json');
const configJSON = fs.readFileSync(configPath, 'utf8');
const config = JSON.parse(configJSON);

//use this identity to query
const appAdmin = "admin"
```

```
33
34 app.post('/login', async (req, res) => {
35
36     let networkObj = await network.connectToNetwork(req.body.username)
       ;
37     if (networkObj.error) {
38         res.send(networkObj.error);
39     }
40
41     var fiveMinutesLater = new Date();
42     fiveMinutesLater.setMinutes(fiveMinutesLater.getMinutes() + 5);
43
44     let args = {
45         "username": req.body.username,
46         "password": req.body.password,
47         "token": Math.random().toString(36).substring(2, 15) + Math.
       random().toString(36).substring(2, 15),
48         "dateTimeLimit": fiveMinutesLater
49     };
50
51     const argsStr = JSON.stringify(args);
52
53     let response = await network.invoke(networkObj, false, 'login', [
       argsStr]);
54
55     res.send(response);
56
57 });
58
59 app.post('/isLoggedIn', async (req, res) => {
60
61     let networkObj = await network.connectToNetwork(appAdmin);
62     if (networkObj.error) {
63         res.send(networkObj.error);
64     }
65
66     let args = {
67         "token": req.body.token,
68         "currentDatetime": new Date()
69     };
70
71     const argsStr = JSON.stringify(args);
72
73     let response = await network.invoke(networkObj, false, '
       loginStatus', [argsStr]);
74
75     res.send(response);
76
77 });
```

```
78
79 //get all assets in world state
80 app.get('/queryAll', async (req, res) => {
81
82     let networkObj = await network.connectToNetwork(appAdmin);
83     let response = await network.invoke(networkObj, true, 'queryAll',
       '');
84     let parsedResponse = await JSON.parse(response);
85     res.send(parsedResponse);
86
87 });
88
89 app.post('/getElectionResults', async (req, res) => {
90
91     let networkObj = await network.connectToNetwork(appAdmin);
92
93     req.body = JSON.stringify(req.body);
94     let args = [req.body];
95
96     let response = await network.invoke(networkObj, true, '
       getResultsByElection', args);
97     let parsedResponse = await JSON.parse(response);
98     res.send(parsedResponse);
99
100 });
101
102 //vote for some candidates. This will increase the vote count for the
       votable objects
103 app.post('/castBallot', async (req, res) => {
104
105     const args = req.body;
106
107     let response = await castBallot(args);
108
109     res.send(response);
110 });
111
112 async function castBallot(args) {
113
114     let networkObj = await network.connectToNetwork(args.username);
115     if (networkObj.error) {
116         res.send(networkObj.error);
117     }
118
119     let rqArgs = {
120         "token": args.token,
121         "electionId": args.electionId,
122         "currentDatetime": new Date(),
123         "picked": args.picked,
```

```
124         "voterEligibleId": args.voterEligibleId,
125     };
126
127     const rqArgsStr = JSON.stringify(rqArgs);
128
129     let response = await network.invoke(networkObj, false, 'castVote',
        [rqArgsStr]);
130
131     return response;
132
133 }
134
135 //query for certain objects within the world state
136 app.post('/getElections', async (req, res) => {
137
138     let networkObj = await network.connectToNetwork(appAdmin);
139     let response = await network.invoke(networkObj, true, '
    queryByObjectType', req.body.selected);
140     let parsedResponse = await JSON.parse(response);
141     let currentDateTime = new Date();
142     let elections = [];
143
144     for (let electionElm of parsedResponse) {
145
146         const start = new Date(electionElm.Record.startDate);
147         const end = new Date(electionElm.Record.endDate);
148         let isElectionRunning = "waiting";
149         if (currentDateTime >= start && currentDateTime < end && !
    electionElm.Record.status) {
150             isElectionRunning = "closed";
151         }
152         else if (currentDateTime < start) {
153             isElectionRunning = "waiting";
154         }
155         else if (electionElm.Record.status) {
156             isElectionRunning = "opened";
157
158         }
159
160         let election = electionElm;
161         election.Record.isRunning = isElectionRunning;
162
163         elections.push(election);
164     }
165
166     res.send(elections);
167
168 });
169
```

```
170
171 //query for certain objects within the world state
172 app.post('/getBallot', async (req, res) => {
173
174     let networkObj = await network.connectToNetwork(req.body.username)
        ;
175
176     req.body = JSON.stringify(req.body);
177     let args = [req.body];
178
179     let response = await network.invoke(networkObj, false, 'getBallot'
        , args);
180     console.log(response);
181
182     res.send(response);
183
184 });
185
186 //query for certain objects within the world state
187 app.post('/queryWithQueryString', async (req, res) => {
188
189     let networkObj = await network.connectToNetwork(appAdmin);
190     let response = await network.invoke(networkObj, true, '
        queryByObjectType', req.body.selected);
191     let parsedResponse = await JSON.parse(response);
192     res.send(parsedResponse);
193
194 });
195
196 app.post('/queryByKey', async (req, res) => {
197
198     let networkObj = await network.connectToNetwork(appAdmin);
199     let response = await network.invoke(networkObj, true, 'readMyAsset
        ', req.body.key);
200     response = JSON.parse(response);
201     if (response.error) {
202         res.send(response.error);
203     } else {
204         res.send(response);
205     }
206 });
207
208
209 app.listen(process.env.PORT || 8081);
210 console.log("Listening port 8081")
```

Listagem II.1: API de suporte à aplicação cliente

```
1 'use strict';
```

```
2
3 const axios = require('axios');
4 const express = require('express');
5 const bodyParser = require('body-parser');
6 const cors = require('cors');
7 const morgan = require('morgan');
8 const util = require('util');
9 const path = require('path');
10 const fs = require('fs');
11
12 let network = require('./fabric/network.js');
13 const { token } = require('morgan');
14
15 const app = express();
16 app.use(morgan('combined'));
17 app.use(bodyParser.json());
18
19 var corsOptions = {
20     origin: '*',
21     optionsSuccessStatus: 200 // some legacy browsers (IE11, various
    SmartTVs) choke on 204
22 }
23
24
25 app.use(cors(corsOptions));
26
27 const configPath = path.join(process.cwd(), '../profiles/
    mychannel_connection_for_nodesdk.json');
28 const configJSON = fs.readFileSync(configPath, 'utf8');
29 const config = JSON.parse(configJSON);
30
31 //use this identity to query
32 const appAdmin = "admin"
33
34 //query for certain objects within the world state
35 app.post('/openElection', async (req, res) => {
36
37     let networkObj = await network.connectToNetwork(appAdmin);
38     req.body = JSON.stringify(req.body);
39     let args = [req.body];
40
41     let response = await network.invoke(networkObj, false, '
    openElection', args);
42     if (response.error) {
43         res.send(response.error);
44     } else {
45
46         res.send(response);
47     }
```

```
48
49 });
50
51 //query for certain objects within the world state
52 app.post('/closeElection', async (req, res) => {
53
54     let networkObj = await network.connectToNetwork(appAdmin);
55     req.body = JSON.stringify(req.body);
56     let args = [req.body];
57
58     let response = await network.invoke(networkObj, false, '
       closeElection', args);
59     if (response.error) {
60         res.send(response.error);
61     } else {
62         res.send(response);
63     }
64
65 });
66
67
68 //query for certain objects within the world state
69 app.post('/createElection', async (req, res) => {
70
71     let networkObj = await network.connectToNetwork(appAdmin);
72
73     req.body = JSON.stringify(req.body);
74     let args = [req.body];
75
76     let response = await network.invoke(networkObj, false, '
       createElection', args);
77     console.log(response);
78
79     response = await network.invoke(networkObj, false, '
       joinVotersToElection', args);
80     console.log(response);
81
82     response = await network.invoke(networkObj, false, '
       joinBallotsToElection', args);
83     console.log(response);
84
85     res.send(response);
86
87 });
88
89 //get voter info, create voter object, and update state with their
       voterId
90 app.post('/registerVoter', async (req, res) => {
91
```

```
92      const response = await registerVoter(req.body);
93
94      if (response.error) {
95          res.send(response.error);
96      } else {
97          res.send(response);
98      }
99
100 });
101
102 async function registerVoter(args) {
103     const voterId = Math.random().toString(36).substring(2, 15) + Math
        .random().toString(36).substring(2, 15);
104     const username = args.username;
105     const password = args.password;
106
107     //first create the identity for the voter and add to wallet
108     let response = await network.registerUser(username);
109     console.log('response from registerVoter: ');
110
111     console.log(response);
112     if (response.error) {
113         return response;
114     }
115
116     console.log("##########################");
117     console.log(args);
118
119     let networkObj = await network.connectToNetwork(username);
120
121     if (networkObj.error) {
122         response.error = networkObj.error;
123         return response;
124     }
125
126     args.voterId = voterId;
127     args = JSON.stringify(args);
128     args = [args];
129     //connect to network and update the state with voterId
130
131     let invokeResponse = await network.invoke(networkObj, false, '
        createVoter', args);
132
133     if (invokeResponse.error) {
134         return invokeResponse;
135     } else {
136
137         let request = {
138             "username": username,
```

```
139             "password": password,
140             "type": "user",
141             "voterId": voterId
142         }
143 
144         let args = [JSON.stringify(request)];
145         //connect to network and update the state with voterId
146 
147         let invokeResponse = await network.invoke(networkObj, false, '
    createUser', args);
148 
149         return invokeResponse.toString();
150     }
151 
152 };
153 
154 //get voter info, create voter object, and update state with their
    voterId
155 app.post('/enrollAdmin', async (req, res) => {
156 
157     //first create the identity for the voter and add to wallet
158     let response = await network.enrollAdmin();
159 
160     let networkObj = await network.connectToNetwork(appAdmin);
161     if (networkObj.error) {
162         res.send(networkObj);
163     }
164 
165     let request = {
166         "username": appAdmin,
167         "password": "adminpw",
168         "type": "admin"
169     }
170 
171     let args = [JSON.stringify(request)];
172     //connect to network and update the state with voterId
173 
174     let invokeResponse = await network.invoke(networkObj, false, '
    createUser', args);
175 
176     if (invokeResponse.error) {
177         res.send(invokeResponse.error);
178     } else {
179         res.send(invokeResponse);
180     }
181 });
182 
183 
184 app.listen(process.env.PORT || 8082);
```

```
185 console.log("Listening port 8082")
```

Listagem II.2: API de administração

```
1  //Import Hyperledger Fabric 1.4 programming model - fabric-network
2  'use strict';
3
4  const FabricCAServices = require('fabric-ca-client');
5  const { Wallets, Gateway, X509WalletMixin } = require('fabric-network'
       );
6  const path = require('path');
7  const fs = require('fs');
8
9  //connect to the config file
10 // const configPath = path.join(process.cwd(), '../profiles/
       mychannel_connection_for_nodesdk.json');
11 // const configJSON = fs.readFileSync(configPath, 'utf8');
12 // const config = JSON.parse(configJSON);
13 // let connection_file = config.connection_file;
14 let userName = "V1";
15 // let gatewayDiscovery = config.gatewayDiscovery;
16 let appAdmin = "admin";
17 let appAdminSecret = "adminpw";
18 let orgConnection = 'org.cne.com';
19 let caConnection = 'ca1.org.cne.com';
20 // let orgMSPID = config.orgMSPID;
21
22 // connect to the connection file
23 const ccpPath = path.join(process.cwd(), '../profiles/
       mychannel_connection_for_nodesdk.json');
24 const ccpJSON = fs.readFileSync(ccpPath, 'utf8');
25 const ccp = JSON.parse(ccpJSON);
26
27 const util = require('util');
28
29 exports.connectToNetwork = async function (userName) {
30
31   const gateway = new Gateway();
32
33   try {
34
35     // Create a new file system based wallet for managing identities.
36     const walletPath = path.join(process.cwd(), '../profiles/vscode/
       wallets', 'org.cne.com');
37     const wallet = await Wallets.newFileSystemWallet(walletPath);
38     console.log(`Wallet path: ${walletPath}`);
39     console.log('userName: ');
40     console.log(userName);
41
```

```
42      console.log('wallet: ');
43      console.log(util.inspect(wallet));
44      console.log('ccp: ');
45      console.log(util.inspect(ccp));
46      // userName = 'V123412';
47      const userIdentity = await wallet.get(userName);
48      if (!userIdentity) {
49        console.log('An identity for the user ' + userName + ' does not
        exist in the wallet');
50        console.log('Run the registerUser.js application before retrying
        ');
51        let response = {};
52        response.error = 'An identity for the user ' + userName + ' does
         not exist in the wallet. Register ' + userName + ' first';
53        return response;
54      }
55
56      await gateway.connect(ccp, { wallet, identity: userName, discovery
        : { enabled: true, asLocalhost: false } });
57
58      // Connect to our local fabric
59      const network = await gateway.getNetwork('mychannel');
60
61      console.log('Connected to mychannel. ');
62      // Get the contract we have installed on the peer
63      const contract = await network.getContract('fabcar');
64
65
66      let networkObj = {
67        contract: contract,
68        network: network,
69        gateway: gateway
70      };
71
72      return networkObj;
73
74    } catch (error) {
75      console.log(`Error processing transaction. ${error}`);
76      console.log(error.stack);
77      let response = {};
78      response.error = error;
79      return response;
80    } finally {
81      console.log('Done connecting to network.');
82      // gateway.disconnect();
83    }
84  };
85
86  exports.invoke = async function (networkObj, isQuery, func, args) {
```

```
87  try {
88    console.log(`isQuery: ${isQuery}, func: ${func}, args: ${args}`);
89
90    if (isQuery === true) {
91
92      if (args) {
93        let response = await networkObj.contract.evaluateTransaction(
    func, args);
94        console.log(response);
95        console.log(`Transaction ${func} with args ${args} has been
    evaluated`);
96
97        await networkObj.gateway.disconnect();
98
99        return response;
100
101     } else {
102
103       let response = await networkObj.contract.evaluateTransaction(
    func);
104       console.log(response);
105       console.log(`Transaction ${func} without args has been
    evaluated`);
106
107       await networkObj.gateway.disconnect();
108
109       return response;
110     }
111   } else {
112     console.log('notQuery');
113     if (args) {
114
115       args = JSON.parse(args[0]);
116
117       args = JSON.stringify(args);
118       console.log(util.inspect(args));
119
120       let response = await networkObj.contract.submitTransaction(
    func, args);
121
122       console.log(response);
123       console.log(`Transaction ${func} with args ${args} has been
    submitted`);
124
125       await networkObj.gateway.disconnect();
126
127       return response;
128
129
```

```
130       } else {
131         let response = await networkObj.contract.submitTransaction(
      func);
132         console.log(response);
133         console.log(`Transaction ${func} with args has been submitted
      `);
134
135         await networkObj.gateway.disconnect();
136
137         return response;
138       }
139     }
140
141   } catch (error) {
142     console.error(`Failed to submit transaction: ${error}`);
143     return error;
144   }
145 };
146
147 exports.registerVoter = async function (voterId, registrarId,
      firstName, lastName) {
148
149   if (!registrarId || !voterId || !firstName || !lastName) {
150     let response = {};
151     response.error = 'Error! You need to fill all fields before you
      can register!';
152     return response;
153   }
154
155   try {
156
157     // Create a new file system based wallet for managing identities.
158     const walletPath = path.join(process.cwd(), 'wallet');
159     const wallet = new FileSystemWallet(walletPath);
160     console.log(`Wallet path: ${walletPath}`);
161     console.log(wallet);
162
163     // Check to see if we've already enrolled the user.
164     const userExists = await wallet.exists(voterId);
165     if (userExists) {
166       let response = {};
167       console.log(`An identity for the user ${voterId} already exists
      in the wallet`);
168       response.error = `Error! An identity for the user ${voterId}
      already exists in the wallet. Please enter
169         a different license number.`;
170       return response;
171     }
172
```

```
173     // Check to see if we've already enrolled the admin user.
174     const adminExists = await wallet.exists(appAdmin);
175     if (!adminExists) {
176       console.log(`An identity for the admin user ${appAdmin} does not
         exist in the wallet`);
177       console.log('Run the enrollAdmin.js application before retrying'
        );
178       let response = {};
179       response.error = `An identity for the admin user ${appAdmin}
        does not exist in the wallet.
180           Run the enrollAdmin.js application before retrying`;
181       return response;
182     }
183
184     // Create a new gateway for connecting to our peer node.
185     const gateway = new Gateway();
186     await gateway.connect(ccp, { wallet, identity: appAdmin, discovery
        : gatewayDiscovery });
187
188     // Get the CA client object from the gateway for interacting with
        the CA.
189     const ca = gateway.getClient().getCertificateAuthority();
190     const adminIdentity = gateway.getCurrentIdentity();
191     console.log(`AdminIdentity: + ${adminIdentity}`);
192
193     // Register the user, enroll the user, and import the new identity
         into the wallet.
194     const secret = await ca.register({ affiliation: '', enrollmentID:
        voterId, role: 'client' }, adminIdentity);
195
196     const enrollment = await ca.enroll({ enrollmentID: voterId,
        enrollmentSecret: secret });
197     const userIdentity = await X509WalletMixin.createIdentity(orgMSPID
        , enrollment.certificate, enrollment.key.toBytes());
198     await wallet.import(voterId, userIdentity);
199     let response = `Successfully registered voter ${firstName} ${
        lastName}. Use voterId ${voterId} to login above.`;
200     console.log(response);
201     return response;
202   } catch (error) {
203     console.error(`Failed to register user + ${voterId} + : ${error}`)
        ;
204     let response = {};
205     response.error = error;
206     return response;
207   }
208 };
209
210
```

```
211 exports.registerUser = async function (userName) {
212   try {
213
214     let mspId = ccp.organizations[orgConnection].mspid;
215
216     // Create a new CA client for interacting with the CA.
217     const caInfo = ccp.certificateAuthorities[caConnection];
218
219     const caURL = caInfo.url
220     const ca = new FabricCAServices(caURL);
221
222     // Create a new file system based wallet for managing identities.
223     const walletPath = path.join(process.cwd(), '../profiles/vscode/
      wallets', orgConnection);
224     const wallet = await Wallets.newFileSystemWallet(walletPath);
225     console.log(`Wallet path: ${walletPath}`);
226
227     // Check to see if we've already enrolled the user.
228     const userIdentity = await wallet.get(userName);
229     if (userIdentity) {
230       console.log(`An identity for the user ${userName} already exists
       in the wallet`);
231       let response = {};
232       response.error = `An identity for the user ${userName} already
      exists in the wallet`;
233       return response;
234     }
235
236     // Check to see if we've already enrolled the admin user.
237     const adminIdentity = await wallet.get(appAdmin);
238     if (!adminIdentity) {
239       console.log(`An identity for the admin user ${appAdmin} does not
       exist in the wallet`);
240       console.log('Run the enrollAdmin.js application before retrying'
      );
241
242       let response = {};
243       response.error = `An identity for the admin user ${appAdmin}
      does not exist in the wallet.
244       Run the enrollAdmin.js application before retrying`;
245       return response;
246     }
247
248     const provider = wallet.getProviderRegistry().getProvider(
      adminIdentity.type);
249     const adminUser = await provider.getUserContext(adminIdentity,
      appAdmin);
250
251     // Register the user, enroll the user, and import the new identity
```

```
     into the wallet.
252     const secret = await ca.register({
253       affiliation: 'org1.department1',
254       enrollmentID: userName,
255       role: 'client'
256     }, adminUser);
257     const enrollment = await ca.enroll({
258       enrollmentID: userName,
259       enrollmentSecret: secret
260     });
261     const x509Identity = {
262       credentials: {
263         certificate: enrollment.certificate,
264         privateKey: enrollment.key.toBytes(),
265       },
266       mspId: mspId,
267       type: 'X.509',
268     };
269     await wallet.put(userName, x509Identity);
270
271     console.log('Successfully registered and enrolled admin user ' +
     userName + ' and imported it into the wallet');
272     let response = {};
273     return response;
274   } catch (error) {
275     console.error(`Failed to register user + ${userName} + : ${error
     }`);
276     let response = {};
277     response.error = error;
278     return response;
279   }
280 };
281
282 exports.enrollAdmin = async function () {
283   try {
284
285     let mspId = ccp.organizations[orgConnection].mspid;
286
287     // Create a new CA client for interacting with the CA.
288     const caInfo = ccp.certificateAuthorities[caConnection];
289     const caTLSCACerts = caInfo.tlsCACerts.pem;
290     const ca = new FabricCAServices(caInfo.url, { trustedRoots:
     caTLSCACerts, verify: false }, caInfo.caName);
291
292     // Create a new file system based wallet for managing identities.
293     const walletPath = path.join(process.cwd(), '../profiles/vscode/
     wallets', orgConnection);
294     const wallet = await Wallets.newFileSystemWallet(walletPath);
295     console.log(`Wallet path: ${walletPath}`);
```

```
296
297     // Check to see if we've already enrolled the admin user.
298     const identity = await wallet.get(appAdmin);
299     if (identity) {
300       console.log('An identity for the admin user "admin" already
        exists in the wallet');
301
302       let response = {};
303       response.error = 'An identity for the admin user "admin" already
         exists in the wallet';
304
305       return response;
306     }
307
308     // Enroll the admin user, and import the new identity into the
        wallet.
309     const enrollment = await ca.enroll({ enrollmentID: appAdmin,
        enrollmentSecret: appAdminSecret });
310     const x509Identity = {
311       credentials: {
312         certificate: enrollment.certificate,
313         privateKey: enrollment.key.toBytes(),
314       },
315       mspId: mspId,
316       type: 'X.509',
317     };
318     await wallet.put(appAdmin, x509Identity);
319
320     let response = 'msg: Successfully enrolled admin user ' + appAdmin
        + ' and imported it into the wallet';
321     console.log(response);
322
323     return response;
324   } catch (error) {
325     console.error(`Failed to enroll admin user ` + ${appAdmin} + : ${
        error}`);
326
327     let response = {};
328     response.error = error;
329     return response;
330
331   }
332 };
```

Listagem II.3: Mecanismo de licação à rede de blockchain.

Apêndice III

Apendice - Chaincode

```
1  /*
2   * Copyright IBM Corp. All Rights Reserved.
3   *
4   * SPDX-License-Identifier: Apache-2.0
5   */
6
7  'use strict';
8
9  const { Contract } = require('fabric-contract-api');
10
11 //import our file which contains our constructors and auxiliary
       function
12 let Ballot = require('./Ballot.js');
13 let User = require('./User.js');
14 let Election = require('./Election.js');
15 let Voter = require('./Voter.js');
16 let VotableItem = require('./VotableItem.js');
17 let VoterEligible = require('./VoterEligible.js');
18
19
20 class VoteContract extends Contract {
21
22     async init(ctx) {
23         console.log('instantiate was called!');
24
25         let response = {}
26         return response;
27
28     }
29
30     /**
31     *
32     * createVoter
33     *
```

```
34      * Creates a voter in the world state, based on the args given.
35      *
36      * @param args.voterId - the Id the voter, used as the key to store
        the voter object
37      * @param args.registrarId - the registrar the voter is registered
        for
38      * @param args.firstName - first name of voter
39      * @param args.lastName - last name of voter
40      * @returns - nothing - but updates the world state with a voter
41      */
42      async createElection(ctx, args) {
43
44          args = JSON.parse(args);
45
46          let response = {};
47
48          let electionExists = await this.myAssetExists(ctx, args.
        electionId);
49
50          if (electionExists) {
51              response.error = `An election with ${args.electionId} id
        already exist`;
52              return response;
53          }
54
55          let electionStartDate = args.electionStartDate;
56          let electionEndDate = args.electionEndDate;
57
58          //create the election
59          let election = await new Election(args.electionId, args.
        electionName, args.electionCountry,
60              args.electionYear, electionStartDate, electionEndDate);
61
62          console.log("Election# " + election.electionId);
63
64          const strElection = JSON.stringify(election);
65          console.log(strElection);
66
67          response = await ctx.stub.putState(election.electionId, Buffer
        .from(strElection));
68
69          console.log("################### Election generated
        successfully");
70
71          return response;
72      }
73
74      async joinVotersToElection(ctx, args) {
75
```

```
76          args = JSON.parse(args);
77
78          let response = {};
79
80          let electionExists = await this.myAssetExists(ctx, args.
        electionId);
81
82          if (!electionExists) {
83              response.error = `The election with ${args.electionId} id
        doesn't exist`;
84              return response;
85          }
86
87          let currVoters = JSON.parse(await this.queryByObjectType(ctx,
        'voter'));
88
89          console.log("############### Voters Existents");
90
91          if (currVoters.length === 0) {
92              response.error = 'No voters. Add voters firstly.';
93              return response;
94          }
95
96          console.log("########### Current Voters");
97          console.log(currVoters);
98
99          // turn voters eligible for cast a vote in this particular
        election
100         for (let voterElm of currVoters) {
101             await this.turnVoterEligible(ctx, JSON.stringify({ voterId
        : voterElm.Key, electionId: args.electionId }));
102         };
103
104         console.log("############## Voters Joined");
105
106         return response;
107     }
108
109     async joinBallotsToElection(ctx, args) {
110
111         args = JSON.parse(args);
112
113         let response = {};
114
115         let electionExists = await this.myAssetExists(ctx, args.
        electionId);
116
117         if (!electionExists) {
```

```
118             response.error = `The election with ${args.electionId} id
    doesn't exist`;
119             return response;
120         }
121 
122         // turn voterconsole.log(args);s eligible for cast a vote in
    this particular election
123         for (let votableElm of args.votableItems) {
124             await this.generateVotableItem(ctx, JSON.stringify({
    votableId: votableElm.votableId,
125                 votableName: votableElm.votableName, description:
    votableElm.description,
126                 electionId: args.electionId }));
127         };
128 
129         console.log("################### Ballots joined successfully")
130 
131         return response;
132     }
133 
134 
135     async closeElection(ctx, args) {
136 
137         args = JSON.parse(args);
138 
139         let response = {};
140 
141         let electionExists = await this.myAssetExists(ctx, args.
    electionId);
142         if (!electionExists) {
143             response.error = `The election with ${args.electionId} id
    doesn't exist`;
144             return response;
145         }
146 
147         let election = await this.readMyAsset(ctx, args.electionId);
148 
149         election.status = false;
150 
151         await ctx.stub.putState(args.electionId, Buffer.from(JSON.
    stringify(election)));
152 
153         console.log("################### Election closed successfully
    ")
154 
155         return response;
156     }
157 
158     async openElection(ctx, args) {
```

```
159
160         args = JSON.parse(args);
161
162         let response = {};
163
164         let electionExists = await this.myAssetExists(ctx, args.
    electionId);
165         if (!electionExists) {
166             response.error = `The election with ${args.electionId} id
    doesn't exist`;
167             return response;
168         }
169
170         let election = await this.readMyAsset(ctx, args.electionId);
171
172         election.status = true;
173
174         await ctx.stub.putState(args.electionId, Buffer.from(JSON.
    stringify(election)));
175
176         console.log("#################### Election opened successfully"
    )
177
178         return response;
179     }
180
181     /**
182     *
183     * generateVotableItem
184     *
185     * Creates a generateVotableItem in the world state.
186     *
187     * @param ctx - the context of the transaction
188     * @param votableItemId
189     * @param electionId
190     * @param description
191     * @returns - nothing
192     */
193     async generateVotableItem(ctx, args) {
194
195         args = JSON.parse(args);
196         console.log(args);
197         console.log(args.votableName);
198
199         //generate votableItem
200         let votableItem = await new VotableItem(args.votableId, args.
    votableName,
201             args.electionId, args.description);
202
```

```
203         console.log("################## votableItem");
204         console.log(votableItem);
205         // //update state with ballot object we just created
206         await ctx.stub.putState(votableItem.votableId, Buffer.from(
    JSON.stringify(votableItem)));
207 
208         console.log("################### PUT votableItem")
209     }
210 
211 
212     // /**
213     // *
214     // * generateBallot
215     // *
216     // * Creates a ballot in the world state, and updates voter ballot
     and castBallot properties.
217     // *
218     // * @param ctx - the context of the transaction
219     // * @param votableItems - The different political parties and
    candidates you can vote for, which are on the ballot.
220     // * @param election - the election we are generating a ballot for
    . All ballots are the same for an election.
221     // * @param voter - the voter object
222     // * @returns - nothing - but updates the world state with a
    ballot for a particular voter object
223     // */
224     async getBallot(ctx, args) {
225 
226         let response = {};
227 
228         args = JSON.parse(args);
229 
230         let queryString = {
231             selector: {
232                 token: args.token
233             }
234         };
235 
236         //check to make sure the election exists
237         let usertokens = JSON.parse(await this.queryWithQueryString(
    ctx, JSON.stringify(queryString)));
238 
239         if (usertokens == null || usertokens.length !== 1) {
240             response.error = `Invalid token`;
241             response.loggedIn = false;
242             return response;
243         }
244 
245         const user = usertokens[0].Record;
```

```
246
247         const voter = await this.getObject(ctx, user.voterId);
248
249         queryString = {
250             selector: {
251                 type: "voter_eligible",
252                 voterId: voter.voterId,
253                 electionId: args.electionId
254             }
255         };
256
257         //check to make sure the election exists
258         let voterEligibles = JSON.parse(await this.
    queryWithQueryString(ctx, JSON.stringify(queryString)));
259
260         if (voterEligibles == null || voterEligibles.length !== 1) {
261             response.error = 'Voter cannot vote';
262             return response;
263         }
264
265         const voterEligible = voterEligibles[0].Record;
266
267         queryString = {
268             selector: {
269                 type: "votableItem",
270                 electionId: voterEligible.electionId
271             }
272         };
273
274         //check to make sure the election exists
275         let votableItems = JSON.parse(await this.queryWithQueryString(
    ctx, JSON.stringify(queryString)));
276
277         if (votableItems == null || votableItems.length == 0) {
278             response.error = 'There are no options to vote';
279             return response;
280         }
281
282         //generate ballot
283         const ballot = await new Ballot(votableItems, voterEligible.
    electionId, voter, voterEligible);
284
285         console.log("#################### Ballot Created")
286
287         return ballot;
288
289     }
290
291
```

```
292     /**
293     *
294     * createVoter
295     *
296     * Creates a voter in the world state, based on the args given.
297     *
298     * @param args.voterId - the Id the voter, used as the key to store
        the voter object
299     * @param args.registrarId - the registrar the voter is registered
        for
300     * @param args.firstName - first name of voter
301     * @param args.lastName - last name of voter
302     * @returns - nothing - but updates the world state with a voter
303     */
304     async createVoter(ctx, args) {
305
306         args = JSON.parse(args);
307
308         //create a new voter
309         let newVoter = await new Voter(args.voterId, args.registrarId,
         args.firstName, args.lastName, args.country);
310
311         //update state with new voter
312         const putVoter = await ctx.stub.putState(newVoter.voterId,
        Buffer.from(JSON.stringify(newVoter)));
313         console.log(putVoter);
314
315         let response = `voter with voterId ${newVoter.voterId} is
        updated in the world state`;
316         console.log(response)
317         return response;
318     }
319
320     async createUser(ctx, args) {
321
322         args = JSON.parse(args);
323
324         const userExists = await this.myAssetExists(ctx, args.username
        );
325         if (userExists) {
326             let response = `User ${args.username} already exists`
327             return response;
328         }
329
330         //create a new voter
331         let newUser = await new User(args.voterId, args.token, args.
        dateTimeLimit, args.username, args.password, args.type);
332
333         //update state with new voter
```

```
334         await ctx.stub.putState(newUser.username, Buffer.from(JSON.
    stringify(newUser)));
335 
336         let response = `User with username ${newUser.username} is
    updated in the world state`;
337         console.log(response)
338         return response;
339     }
340 
341 
342     /**
343      *
344      * deleteMyAsset
345      *
346      * Deletes a key-value pair from the world state, based on the key
     given.
347      *
348      * @param myAssetId - the key of the asset to delete
349      * @returns - nothing - but deletes the value in the world state
350      */
351     async deleteMyAsset(ctx, myAssetId) {
352 
353         const exists = await this.myAssetExists(ctx, myAssetId);
354         if (!exists) {
355             throw new Error(`The my asset ${myAssetId} does not exist
    `);
356         }
357 
358         await ctx.stub.deleteState(myAssetId);
359 
360     }
361 
362     /**
363      *
364      * readMyAsset
365      *
366      * Reads a key-value pair from the world state, based on the key
    given.
367      *
368      * @param myAssetId - the key of the asset to read
369      * @returns - nothing - but reads the value in the world state
370      */
371     async readMyAsset(ctx, myAssetId) {
372 
373         const exists = await this.myAssetExists(ctx, myAssetId);
374 
375         if (!exists) {
376             // throw new Error(`The my asset ${myAssetId} does not
    exist`);
```

```
377             let response = {};
378             response.error = `The my asset ${myAssetId} does not exist
    `;
379             return response;
380         }
381
382         const buffer = await ctx.stub.getState(myAssetId);
383         const asset = JSON.parse(buffer.toString());
384         return asset;
385     }
386
387
388
389     /**
390      *
391      * myAssetExists
392      *
393      * Checks to see if a key exists in the world state.
394      * @param myAssetId - the key of the asset to read
395      * @returns boolean indicating if the asset exists or not.
396      */
397     async myAssetExists(ctx, myAssetId) {
398
399         const buffer = await ctx.stub.getState(myAssetId);
400         return (!!buffer && buffer.length > 0);
401
402     }
403
404     /**
405      *
406      * turnVoterEligible
407      *
408      * First to checks that a particular voterId and the electionId
    exists, and then
409      * checks if the voter is eligible for this election, if yes
    create a state in the world state.
410      *
411      * @param electionId - the electionId of the election we want to
    vote in
412      * @param voterId - the voterId of the voter that wants to vote
413      * @returns response status.
414      */
415     async turnVoterEligible(ctx, args) {
416         args = JSON.parse(args);
417
418         //make sure we have an election
419         let electionAsBytes = await ctx.stub.getState(args.electionId)
    ;
420         let election = await JSON.parse(electionAsBytes);
```

```
421         let voterAsBytes = await ctx.stub.getState(args.voterId);
422         let voter = await JSON.parse(voterAsBytes);
423 
424         if (voter.country == null || voter.country != election.country
    ) {
425             return;
426         }
427 
428         let voterEligible = await new VoterEligible(voter.voterId,
    election.electionId);
429 
430         let voterEligibleExists = await this.myAssetExists(ctx,
    voterEligible.voterEligibleId);
431         if (voterEligibleExists) {
432             return;
433         }
434 
435         //update the state with the new vote count
436         let result = await ctx.stub.putState(voterEligible.
    voterEligibleId, Buffer.from(JSON.stringify(voterEligible)));
437         console.log(result);
438 
439         return result;
440     }
441 
442 
443     async login(ctx, args) {
444         args = JSON.parse(args);
445         let response = {};
446 
447         //check to make sure the election exists
448         let userExists = await this.myAssetExists(ctx, args.username);
449         if (!userExists) {
450             response.error = `User ${args.username} does not exist!`;
451             return response;
452         }
453 
454         //make sure we have an election
455         let user = await this.getObject(ctx, args.username);
456 
457         if (user.password != args.password) {
458 
459             response.error = `User password is invalid `;
460             return response;
461         }
462 
463         let dateTimeLimit = await Date.parse(args.dateTimeLimit);
464 
465         user.dateTimeLimit = dateTimeLimit;
```

```
466         user.token = args.token;
467
468         const result = await ctx.stub.putState(user.username, Buffer.
    from(JSON.stringify(user)));
469         console.log(result);
470
471         user.password = "";
472
473         response.user = user;
474
475         return response;
476
477     }
478
479
480     async loginStatus(ctx, args) {
481         args = JSON.parse(args);
482         let response = {};
483
484         let queryString = {
485             selector: {
486                 token: args.token
487             }
488         };
489
490         //check to make sure the election exists
491         let usertokens = JSON.parse(await this.queryWithQueryString(
    ctx, JSON.stringify(queryString)));
492
493         if (usertokens == null || usertokens.length !== 1) {
494             response.error = `Invalid token`;
495             response.loggedIn = false;
496             return response;
497         }
498
499         let user = usertokens[0].Record;
500
501         const dateA = new Date(user.dateTimeLimit);
502         const dateB = new Date(args.currentDatetime);
503
504         if (dateA < dateB) {
505
506             response.error = `Invalid token - Expired`;
507             response.loggedIn = false;
508             return response;
509         }
510
511         response.loggedIn = true;
512
```

```
513         console.log("############## Login status tested successfully");
514
515         return response;
516     }
517
518     /**
519      *
520      * castVote
521      *
522      * First to checks that a particular voterId has not voted before,
     and then
523      * checks if it is a valid election time, and if it is, we
    increment the
524      * count of the political party that was picked by the voter and
    update
525      * the world state.
526      *
527      * @param electionId - the electionId of the election we want to
    vote in
528      * @param voterId - the voterId of the voter that wants to vote
529      * @param votableId - the Id of the candidate the voter has
    selected.
530      * @returns an array which has the winning briefs of the ballot.
531      */
532     async castVote(ctx, args) {
533         args = JSON.parse(args);
534         let response = {};
535         //get the political party the voter voted for, also the key
536         let votableId = args.picked;
537
538         let voterEligibleId = args.voterEligibleId;
539
540         console.log("############## Check Login Status");
541         const loginStatus = await this.loginStatus(ctx, JSON.stringify
    (args));
542
543         if (loginStatus.loggedIn == false) {
544             return loginStatus;
545         }
546
547         console.log("############## Check voterEligibleId");
548
549         //check to make sure the election exists
550         let voterEligibleExists = await this.myAssetExists(ctx,
    voterEligibleId);
551         if (voterEligibleExists) {
552
553             //make sure we have an election
```

```
554             let voterEligibleAsBytes = await ctx.stub.getState(
    voterEligibleId);
555             let voterEligible = await JSON.parse(voterEligibleAsBytes)
    ;
556
557             console.log("############ Get voterEligibleId");
558
559             if (voterEligible.ballotCasted) {
560
561                 response.error = 'this voter has already cast this
    ballot!';
562                 return response;
563             }
564
565             console.log("############ Get election");
566
567             const election = await this.getObject(ctx, voterEligible.
    electionId)
568
569             console.log(`BEFORE #-> electionStart ${election.startDate
    } ############ electionEnd ${election.endDate} ###### currentTime
    ${args.currentDatetime}`);
570
571             //check the date of the election, to make sure the
    election is still open
572             let currentTime = new Date(args.currentDatetime);
573
574             //parse date objects
575             let electionStart = new Date(election.startDate);
576             let electionEnd = new Date(election.endDate);
577
578             console.log(`AFTER #-> electionStart ${electionStart}
    ############ electionEnd ${electionEnd} ###### currentTime ${
    currentTime}`);
579
580             console.log("############ Check election time range");
581             //only allow vote if the election has started
582             if (currentTime >= electionStart && currentTime <
    electionEnd) {
583
584                 console.log("############ Get Votable Item");
585
586                 //check to make sure the votable item exists
587                 let votableItem = await this.getObject(ctx, votableId)
    ;
588                 if (votableItem) {
589                     await votableItem.count++;
590
591                     console.log("############ Increment vote");
```

```
592
593                     voterEligible.ballotCasted = true;
594                     voterEligible.ballotCastedDate = currentTime;
595
596                     console.log("############ Save update count");
597
598                     let result = await ctx.stub.putState(votableId,
    Buffer.from(JSON.stringify(votableItem)));
599                     console.log(result);
600
601                     console.log("############ Save update
    voterEligible");
602
603                     result = await ctx.stub.putState(voterEligible.
    voterEligibleId, Buffer.from(JSON.stringify(voterEligible)));
604                     console.log(result);
605                 } else {
606                     response.error = 'VotableId does not exist!';
607                     return response;
608                 }
609
610                 console.log("############ Vote casted");
611
612                 console.log(response);
613                 return response;
614
615             } else {
616                 response.error = 'the election is not open now!';
617                 return response;
618             }
619
620         } else {
621             response.error = 'the election or the voter does not exist
    !';
622             return response;
623         }
624     }
625
626
627     async getObject(ctx, key) {
628         let objExists = await this.myAssetExists(ctx, key);
629         if (objExists) {
630             let objAsBytes = await ctx.stub.getState(key);
631             let objJson = await JSON.parse(objAsBytes);
632
633             return objJson;
634         }
635
636         return;
```

```
637     }
638
639     async getResultsByElection(ctx, args) {
640
641         args = JSON.parse(args);
642         let response = {};
643
644         let queryString = {
645             selector: {
646                 electionId: args.electionId,
647                 type: "votableItem"
648             }
649         };
650
651         let queryResults = await this.queryWithQueryString(ctx, JSON.
    stringify(queryString));
652
653         response.results = JSON.parse(queryResults);
654
655         return response;
656
657     }
658
659     /**
660      * Query and return all key value pairs in the world state.
661      *
662      * @param {Context} ctx the transaction context
663      * @returns - all key-value pairs in the world state
664     */
665     async queryAll(ctx) {
666
667         let queryString = {
668             selector: {}
669         };
670
671         let queryResults = await this.queryWithQueryString(ctx, JSON.
    stringify(queryString));
672         return queryResults;
673
674     }
675
676
677     /**
678    * Query by the main objects in this app: ballot, election,
       votableItem, and Voter.
679    * Return all key-value pairs of a given type.
680    *
681    * @param {Context} ctx the transaction context
```

```
682   * @param {String} objectType the type of the object - should be
        either ballot, election, votableItem, or Voter
683   */
684     async queryByObjectType(ctx, objectType) {
685
686         let queryString = {
687             selector: {
688                 type: objectType
689             }
690         };
691
692         let queryResults = await this.queryWithQueryString(ctx, JSON.
        stringify(queryString));
693         return queryResults;
694
695     }
696
697     /**
698     * Evaluate a queryString
699     *
700     * @param {Context} ctx the transaction context
701     * @param {String} queryString the query string to be evaluated
702     */
703     async queryWithQueryString(ctx, queryString) {
704
705         console.log('query String');
706         console.log(JSON.stringify(queryString));
707
708         let resultsIterator = await ctx.stub.getQueryResult(
        queryString);
709
710         let allResults = [];
711
712         // eslint-disable-next-line no-constant-condition
713         while (true) {
714             let res = await resultsIterator.next();
715
716             if (res.value && res.value.value.toString()) {
717                 let jsonRes = {};
718
719                 console.log(res.value.value.toString('utf8'));
720
721                 jsonRes.Key = res.value.key;
722
723                 try {
724                     jsonRes.Record = JSON.parse(res.value.value.
                    toString('utf8'));
725                 } catch (err) {
726                     console.log(err);
```

```
727                     jsonRes.Record = res.value.value.toString('utf8');
728                 }
729
730                 allResults.push(jsonRes);
731             }
732             if (res.done) {
733                 console.log('end of data');
734                 await resultsIterator.close();
735                 console.info(allResults);
736                 console.log(JSON.stringify(allResults));
737                 return JSON.stringify(allResults);
738             }
739         }
740     }
741
742 }
743
744 module.exports = VoteContract;
```

Listagem III.1: Chaincode

```
1  'use strict';
2
3  class Ballot {
4
5    /**
6     *
7     * Ballot
8     *
9     * Constructor for a Ballot object. This is what the point of the
       application is - create
10    * ballots, have a voter fill them out, and then tally the ballots.
11    *
12    * @param items - an array of choices
13    * @param election - what election you are making ballots for
14    * @param voterId - the unique Id which corresponds to a registered
       voter
15    * @returns - registrar object
16    */
17   constructor(items, electionId, voter, voterEligible) {
18
19     this.votableItems = items;
20     this.electionId = electionId;
21     this.voter = voter;
22     this.voterEligible = voterEligible;
23     this.type = 'ballot';
24     if (this.__isContract) {
25       delete this.__isContract;
26     }
```

```
27      if (this.name) {
28        delete this.name;
29      }
30      return this;
31
32
33    }
34  }
35  module.exports = Ballot;
```

Listagem III.2: Ballot

```
1   'use strict';
2
3   class Election {
4
5     /**
6      *
7      * validateElection
8      *
9      * check for valid election, cross check with government. Don't want
         duplicate
10     * elections.
11     *
12     * @param electionId - an array of choices
13     * @returns - yes if valid Voter, no if invalid
14     */
15    async validateElection(electionId) {
16
17      //registrarId error checking here, i.e. check if valid drivers
        License, or state ID
18      if (electionId) {
19        return true;
20      } else {
21        return false;
22      }
23    }
24    /**
25     *
26     * Election
27     *
28     * Constructor for an Election object. Specifies start and end date.
29     *
30     * @param items - an array of choices
31     * @param election - what election you are making ballots for
32     * @param voterId - the unique Id which corresponds to a registered
        voter
33     * @returns - registrar object
34     */
```

```
35    constructor(electionId, name, country, year, startDate, endDate) {
36
37      this.electionId = electionId;
38
39      if (this.validateElection(this.electionId)) {
40
41        //create the election object
42        this.name = name;
43        this.country = country;
44        this.year = year;
45        this.startDate = startDate;
46        this.endDate = endDate;
47        this.status = false
48        this.type = 'election';
49        if (this.__isContract) {
50          delete this.__isContract;
51        }
52        return this;
53
54      } else {
55        throw new Error('not a valid election!');
56      }
57
58    }
59
60  }
61  module.exports = Election;
```

Listagem III.3: Election

```
1   'use strict';
2
3   class User {
4     /**
5      *
6      * Voter
7      *
8      * Constructor for a Voter object. Voter has a voterId and registrar
         that the
9      * voter is .
10     *
11     * @param items - an array of choices
12     * @param election - what election you are making ballots for
13     * @param voterId - the unique Id which corresponds to a registered
        voter
14     * @returns - registrar object
15     */
16    constructor(voterId, token, dateTimeLimit, username, password, type)
        {
```

```
17
18      this.voterId = voterId;
19      this.token = token;
20      this.dateTimeLimit = dateTimeLimit;
21      this.username = username;
22      this.password = password;
23      this.type = type;
24      if (this.__isContract) {
25        delete this.__isContract;
26      }
27      if (this.name) {
28        delete this.name;
29      }
30      return this;
31
32
33    }
34  }
35  module.exports = User;
```

Listagem III.4: User

```
1  /* eslint-disable indent */
2  'use strict';
3
4  class VotableItem {
5
6    /**
7   *
8   * VotableItem
9   *
10  * Constructor for a VotableItem object. These will eventually be
      placed on the
11  * ballot.
12  *
13  * @param votableId - the Id of the votableItem
14  * @param description - the description of the votableItem
15  * @param voterId - the unique Id which corresponds to a registered
      voter
16  * @returns - registrar object
17  */
18    constructor(votableId, votableName, electionId, description) {
19
20      this.votableId = votableId;
21      this.votableName = votableName;
22      this.description = description;
23      this.electionId = electionId;
24      this.count = 0;
25      this.type = 'votableItem';
```

```
26      if (this.__isContract) {
27        delete this.__isContract;
28      }
29      return this;
30
31    }
32  }
33  module.exports = VotableItem;
```

Listagem III.5: VotableItem

```
1   'use strict';
2
3   class Voter {
4     /**
5      *
6      * Voter
7      *
8      * Constructor for a Voter object. Voter has a voterId and registrar
         that the
9      * voter is .
10     *
11     * @param items - an array of choices
12     * @param election - what election you are making ballots for
13     * @param voterId - the unique Id which corresponds to a registered
       voter
14     * @returns - registrar object
15     */
16    constructor(voterId, registrarId, firstName, lastName, country) {
17
18      if (this.validateVoter(voterId) && this.validateRegistrar(
       registrarId)) {
19
20        this.voterId = voterId;
21        this.registrarId = registrarId;
22        this.firstName = firstName;
23        this.lastName = lastName;
24        this.country = country;
25        this.type = 'voter';
26        if (this.__isContract) {
27          delete this.__isContract;
28        }
29        if (this.name) {
30          delete this.name;
31        }
32        return this;
33
34      } else if (!this.validateVoter(voterId)) {
35        throw new Error('the voterId is not valid.');
```

```
36      } else {
37        throw new Error('the registrarId is not valid.');
38      }
39
40    }
41
42    /**
43     *
44     * validateVoter
45     *
46     * check for valid ID card - stateID or drivers License.
47     *
48     * @param voterId - an array of choices
49     * @returns - yes if valid Voter, no if invalid
50     */
51    async validateVoter(voterId) {
52      //VoterId error checking here, i.e. check if valid drivers License
       , or state ID
53      if (voterId) {
54        return true;
55      } else {
56        return false;
57      }
58    }
59
60    /**
61     *
62     * validateRegistrar
63     *
64     * check for valid registrarId, should be cross checked with
       government
65     *
66     * @param voterId - an array of choices
67     * @returns - yes if valid Voter, no if invalid
68     */
69    async validateRegistrar(registrarId) {
70
71      //registrarId error checking here, i.e. check if valid drivers
       License, or state ID
72      if (registrarId) {
73        return true;
74      } else {
75        return false;
76      }
77    }
78
79  }
80  module.exports = Voter;
```

Listagem III.6: Voter

```
'use strict';

class VoterEligible {
  /**
   *
   * VoterEligible
   *
   * Constructor for a VoterEligible object. VoterEligible has a voterId and an electionId that the
   * voter is .
   *
   * @param electionId - what election you are making ballots for
   * @param voterId - the unique Id which corresponds to a registered voter
   * @returns - registrar object
   */
  constructor(voterId, electionId) {

    this.voterEligibleId = `${voterId}${electionId}`;
    this.voterId = voterId;
    this.electionId = electionId;
    this.ballotCasted = false;
    this.ballotCastedDate = null;
    this.type = 'voter_eligible';
    if (this.__isContract) {
      delete this.__isContract;
    }
    if (this.name) {
      delete this.name;
    }
    return this;
  }

}
module.exports = VoterEligible;
```

Listagem III.7: VoterEligible