UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

LUIZA DOMINGUES MEDEIROS

Rastreabilidade do ENEM na Jornada do Estudante

FLORIANÓPOLIS

2024

LUIZA DOMINGUES MEDEIROS

Rastreabilidade do ENEM na Jornada do Estudante

Trabalho de Conclusão do Curso de Graduação em Sistemas de Informação, do Departamento de Informática e Estatística, do Centro Tecnológico da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Jean Everson Martina, Dr.

FLORIANÓPOLIS
2024

RESUMO

A tecnologia de *blockchain* ganha cada vez mais atenção em inúmeras áreas diferentes, que buscam entender a possibilidade de aplicação da tecnologia para cada contexto, uma vez que a *blockchain* permite o armazenamento de informações de forma imutável, distribuída, e descentralizada, registrando cada transação que acontece de forma altamente confiável. Atualmente, as aplicações mais conhecidas de *blockchain* encontram-se no mercado de criptomoedas, contanto, entende-se que existem benefícios intrínsecos às *blockchain* os quais as fazem vantajosas de serem aplicadas em outras áreas. No Brasil, muitos alunos que procuram financiamento ou até mesmo ingresso em uma Instituição de Ensino Superior (IES) deverão realizar o Exame Nacional do Ensino Médio (ENEM), uma vez que este é critério de seleção dos principais programas públicos de ingresso em IES. Assim, o sucesso e confiabilidade desses processos dependem do êxito e integridade dos resultados obtidos a partir do ENEM. Nesse contexto, propõe-se, neste trabalho, a aplicação de uma *blockchain* para fazer o registro e acompanhamento de movimentações de candidatos do ENEM, visando garantir, por meio das características das *blockchains*, maior rastreabilidade e segurança no processo.

Palavras-chave: Blockchain; Jornada do Estudante; Hyperledger Fabric; Contratos Inteligentes.

ABSTRACT

Blockchain technology enables the immutable, distributed, and decentralized storage of information, recording each transaction highly reliably. Currently, the most well-known applications of blockchain are found in the cryptocurrency market; however, it is understood that there are intrinsic benefits to blockchain that make it advantageous to be applied in other areas. In Brazil, many students seeking financing or admission to Higher Education Institutions must take the National High School Exam (ENEM), as it is a selection criterion for the main public admission programs in HEIs. Thus, the success and reliability of these processes depend on the success and integrity of the results obtained from the ENEM. In this context, this work proposes the application of blockchain to record and track the movements of candidates of the ENEM, aiming to ensure greater traceability and security in the process through the characteristics of blockchains.

Keywords: Blockchain; Hyperledger Fabric; Smart Contracts.

SUMÁRIO

1. INTRODUÇÃO

Ao iniciar-se o processo do uso pessoal de computadores, o criptógrafo Stuart Haber e o físico Scott Stornetta questionaram o quão confiável seriam as informações salvas nas máquinas ao longo dos anos se era tão simples editar os conteúdos de um arquivo em um computador (WHITAKER, 2019). Além disso, surgiu, entre a dupla, a preocupação do quanto poderia se confiar em uma autoridade central que fosse responsável pelo registro dessas informações. Então, nasceu a ideia de projetar um sistema que fosse criptografado e registrado, com o *timestamp* desses registros interligados de tal modo que seria impossível manipular um registro sem causar um transtorno na rede inteira. Nesse contexto, cada bloco de transações seria ligado um ao outro, internamente, com várias cópias distribuídas da rede dispensando a necessidade de se confiar em um administrador central para fazer a salvaguarda das informações. Assim foi criado o artigo "*How to Time-Stamp a Digital Document*" (HABER; STORNETTA, 1991), com a ideia que serviria como ponto de partida para as *blockchains* como elas são hoje.

Em 2003, ainda segundo Whitaker (2019), Haber e Stornetta fundaram a empresa *Surety,* com o intuito de oferecer a cientistas um local de armazenamento seguro para suas anotações e pesquisas. A *Surety* publicava, semanalmente, um código alfanumérico no *The New York Times* (WHITAKER, 2019), visando oferecer aos cientistas a opção de conferirem, caso desejassem, se os arquivos da rede de *blockchain* da empresa haviam sido violados. Atualmente, o código da empresa segue sendo publicado no *The New York Times*, tornando a *blockchain* da Surety a mais antiga do mundo.

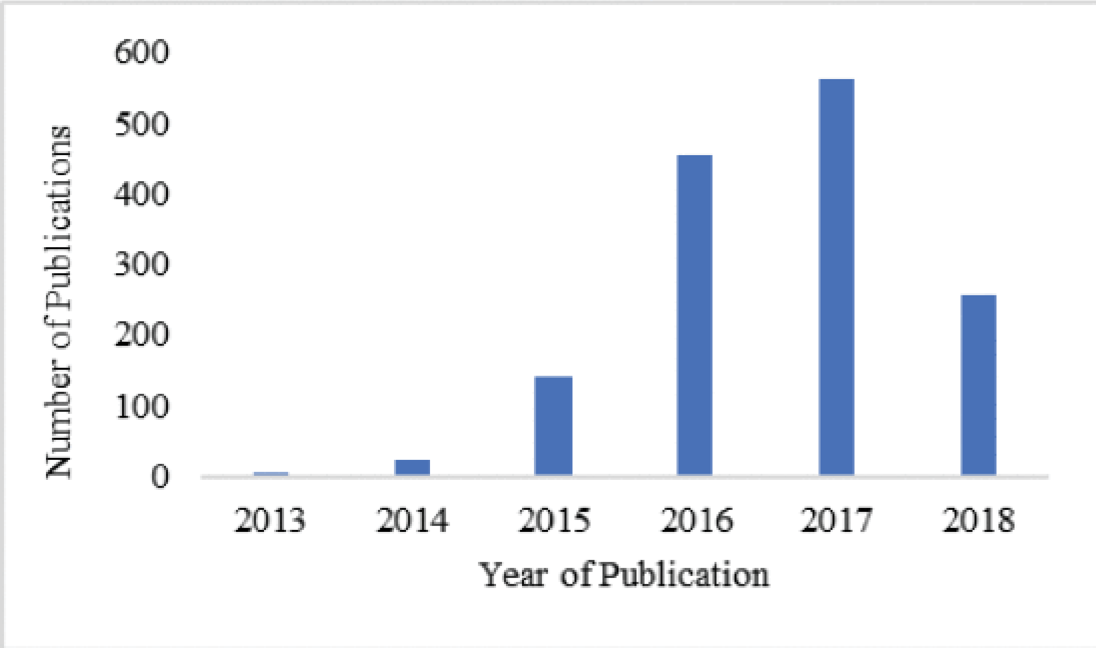
No entanto, o conceito de *blockchain* ganhou maior visibilidade em 2009, quando a estrutura de Haber e Stornetta foi somada a um conceito de incentivo financeiro por Satoshi Nakamoto, propondo, assim, a aplicação mais conhecida de *blockchain* na atualidade: o Bitcoin (WHITAKER, 2019). A principal diferença que Satoshi trouxe com o Bitcoin foi o conceito de que cada participante poderia ganhar moedas - *bitcoins* - como recompensa por resolver problemas matemáticos ligados às transações dos blocos. Em 2014, Vitalik Buterin apresentou ao mundo o protocolo de Ethereum, que serviria como uma estrutura a base de contratos inteligentes que permitia tokenização, generalizando parte do código do Bitcoin para que o mesmo pudesse rodar diversos tipos de programas (WHITAKER, 2019). Com o tempo, alguns desses contratos inteligentes tornaram-se padronizados - tornando dois *tokens* intercambiáveis, por exemplo.

Desse modo, a tecnologia de *blockchain* baseia-se na descentralização e imutabilidade, apresentando potencial transformador em diversos contextos, recebendo destaque como “uma das tecnologias mais revolucionárias do mundo” (EXAME, 2022). A possibilidade de se adotar uma tecnologia que mantém um registro de transações sem que haja uma autoridade centralizadora se dá uma vez que dispensa a necessidade de se confiar nessa autoridade - assim como Haber e Stornetta visavam. Isso acontece visto que, caso o centralizador viesse a perceber vantagem em manipular ou alterar algum registro, o mesmo poderia fazê-lo. A dispensa da necessidade de se confiar em tal centralizador é garantido, por exemplo, na tecnologia *blockchain*, justamente por não contar com um centralizador de informações. Além disso, por contar com registros distribuídos das transações, as *blockchains* tornam-se auditáveis por todos que participam da rede, garantindo transparência, promovendo a agilidade no processamento de transações e trocas, e prevenindo fraudes, uma vez que alterações dependem da aprovação da rede como um todo (JAOUDE; SAADÉ, 2019).

Logo, a tecnologia tem recebido muita atenção de inúmeras frentes diferentes, recebendo um pico de interesse nas pesquisas acadêmicas a partir de 2016, segundo Jaoude e Saadé, visando entender onde como ela pode ser utilizada em determinados contextos - do mercado bancário até gerenciamento de base de dados no contexto da área da saúde (JAOUDE; SAADÉ, 2019).

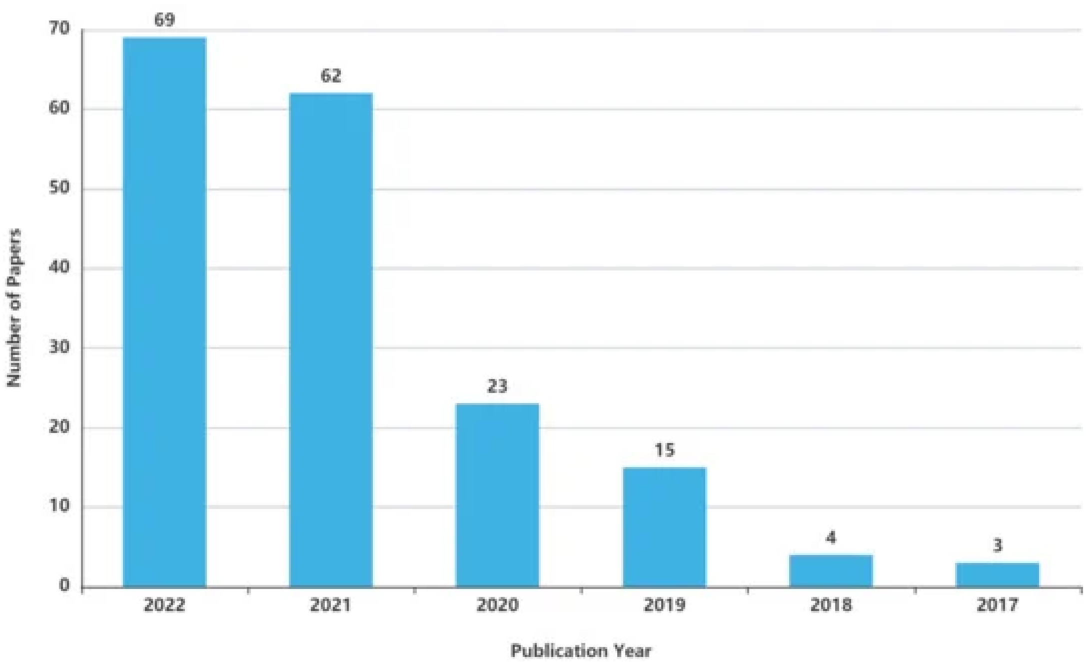
Com o intuito de melhor entender as pesquisas feitas referentes ao tema, em 2019, Jaoude e Saadé realizaram uma revisão sistêmica da literatura. O resumo do resultado pode ser conferido na figura 1. Já em 2022, Liu, Si, e Kang realizaram uma pesquisa similar, buscando entender a publicação de trabalhos que relacionavam *blockchain* com a cadeia de mantimentos. O resultado pode ser conferido na figura 2.

Figura 1 - Número de publicações sobre *blockchain* (2013 - 2018)



Fonte: Jaoude e Saadé (2019)

Figura 2 - Número de publicações sobre “blockchain” e “supply chain” por ano (2017 - 2022)



Fonte: Liu, Si, e Kang (2022)

Conforme pode-se perceber, através das figuras, há um interesse crescente expressivo pela tecnologia nos últimos anos, e uma crescente demanda para entender as diferentes aplicações da tecnologia.

O Exame Nacional de Ensino Médio (ENEM) tornou-se, em 2009, um mecanismo de ingresso à educação superior no Brasil, influenciando o acesso a inúmeras oportunidades acadêmicas, uma vez que sua nota passou a ser utilizada para diversos programas de admissão ou financiamento estudantil do governo brasileiro (Ministério da Educação, 2023). Os principais processos de seleção que o governo oferece aos estudantes, como o Programa Universidade Para Todos (Prouni), Sistema de Seleção Unificado (SiSU), e Fundo de Financiamento Estudantil (Fies), todos têm como requisito a nota do ENEM. Nesse sentido, todo ano, a prova do ENEM é feita por milhões de estudantes pelo país, com 3.9 milhões de inscritos em 2023 (Assessoria de Comunicação Social do Inep, 2023). Além disso, conforme explicitado na página online do ENEM, o Ministério da Educação (MEC) utiliza as notas resultantes do exame em cada ano para desenvolvimento de estudos e indicadores institucionais. Assim, o exame destaca-se como "a maior porta de entrada para o ensino superior público e particular do Brasil" (BATISTA, 2018), com um impacto notório tanto para os estudantes brasileiros quanto para o próprio acompanhamento do MEC em relação ao estado do ensino no país.

Estabelecida a importância do ENEM para o país, entende-se que é crucial transparência, confiabilidade, rastreabilidade, e segurança nos processos relacionados ao mesmo, a fim de garantir assertividade nos procedimentos de admissão e seleção dos estudantes brasileiros. Hoje, já existe, no Projeto do Jornada do Estudante, uma série de contratos implementados e/ou sendo desenvolvidos, voltados para o acompanhamento da emissão do diploma do estudante, e do seu histórico escolar, entre outras informações, utilizando blockchain. Então, enxerga-se uma oportunidade de trazer o processo do ENEM ao projeto, aproveitando a estrutura já existente, para realizar uma extensão da mesma e concretizar a inclusão do ENEM, e, posteriormente, os processos seletivos que decorrem dele, à tecnologia existente. Assim, entende-se que no domínio das admissões acadêmicas, a utilização de *blockchain,* que já está sendo utilizada no Jornada, demonstra uma solução promissora para aprimorar a credibilidade e registro das notas e movimentações do ENEM, além do rastreio dos candidatos ao longo do processo da prova.

1.1. OBJETIVOS

Visando garantir maior clareza na organização do escopo do trabalho que será desenvolvido, esta seção foi separada em: 1.1.1 Objetivos gerais, que busca apresentar o escopo de forma mais generalista; 1.1.2 Objetivos específicos, que pontua de forma mais aprofundada os resultados que se espera obter através do desenvolvimento deste trabalho.

1.1.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver uma *chaincode* para fazer a para fazer a rastreabilidade do candidato do ENEM. A *chaincode* irá acompanhar e registrar todo o processo, de início ao fim, do candidato do ENEM, contemplando, assim, a etapa de inscrição no ENEM.

1.1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Definir os requisitos e funcionalidades da rede *blockchain*, analisando as necessidades da aplicação;
- Projetar um protótipo de arquitetura para uma rede *blockchain*;
- Implementar um contrato que armazenará e acompanhará o processo de inscrição de cada estudante no ENEM, até o momento de emissão do resultado da prova, na estrutura atual do Jornada do Estudante..

1.2. METODOLOGIA

A fim de alcançar os resultados esperados com este trabalho, foi adotado uma combinação de metodologias de pesquisa de acordo com o respectivo objetivo de cada etapa.

Inicialmente, foi feita uma metodologia exploratória, com foco em conhecer o histórico e garantir o entendimento teórico das tecnologias que serão utilizadas para a construção da rede *blockchain*, além de compreender a posição das pesquisas atuais sobre os assuntos que serão tratados ao longo deste trabalho. Assim, a primeira área de estudo foi referente a *blockchain*, no intuito de entender a teoria e os benefícios e desvantagens da tecnologia como um todo, além de buscar a compreensão e seus diferentes tipos, que serão abordados na fundamentação teórica, como *blockchain* pública, privada, contratos inteligentes, e suas possíveis arquiteturas e diferentes vantagens das mesmas. Estudos dessa etapa também buscaram garantir entender os processos que utilizam a nota do ENEM como critério de seleção, com o SiSU, Prouni, entre outros.

De acordo com o problema identificado, de falta de rastreabilidade no registro de notas do ENEM dos estudantes, e seus processos seletivos ao longo dos anos, busca-se projetar um contrato inteligente que registre o processo do candidato na prova do ENEM.

Assim, deve-se entender os dados envolvidos no processo, conforme detalhado na seção 3.1, para então definir as regras de negócio, e implementar o contrato. O código do contrato será comentado na seção 3.4, explicando a implementação, além de abordar as dificuldades e desafios encontrados ao longo do desenvolvimento, na seção 3.5. Na seção 4, alguns testes serão demonstrados, para demonstrar a implementação das regras de negócio. Por fim, na seção 5, apresentam-se as considerações finais e os trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os assuntos que formam a base teórica necessária para entender *blockchain* e o funcionamento da tecnologia, visando, assim, facilitar o entendimento daquilo que é proposto neste trabalho. Para organizar melhor os conteúdos, será dividido em duas partes principais.

2.1 CONTEÚDOS TÉCNICOS

Na primeira parte da fundamentação teórica, tecnologias e conceitos técnicos necessários para o entendimento e desenvolvimento do trabalho serão abordados.

2.1.1 Função hash criptográfica

A função hash refere-se a uma técnica de criptografia que converte dados, na forma de uma sequência de caracteres, em uma cadeia alfanumérica de tamanho fixo. Quando essa função atende a determinados requisitos, ela é classificada como uma função de criptografia hash. O algoritmo de hash criptográfico é amplamente reconhecido como "uma das ferramentas mais utilizadas no campo da criptografia" (SOBTI; GEETHA, 2012, p. 462). Ela é empregada em diversas aplicações, incluindo assinaturas digitais, verificação de autenticidade e em redes *blockchain*, como será discutido mais adiante.

Segundo Merkle (1989), os requisitos a serem satisfeitos para que uma função seja classificada como uma função hash de sentido único, seja F uma função, são:

1. F pode ser utilizado em um entrada de qualquer tamanho;
2. F irá produzir um resultado de tamanho fixo;
3. Dado F e x, é fácil calcular F(x) - ou seja, independente da entrada, não irá demandar muito do poder computacional de uma máquina o cálculo do resultado da função;
4. Dado F, é computacionalmente inviável achar um par x, x’, dado que $x \neq x'$ e $F(x)=F(x')$.

Ainda segundo Sobti e Geetha (2012), outros fatores característicos que reforçam a segurança de uma função criptográfica hash são:

1. Forte efeito avalanche: uma mudança pequena na entrada gera um resultado com diferenças significativas. Isso auxilia na conferência de integridade dos dados, uma vez que alterações geram resultados bastante perceptíveis;

2. Determinismo: para uma entrada x, F(x) terá sempre o mesmo resultado para aquela função F, independente de quantas vezes a função seja aplicada.
3. Resistência a colisões: Dado F, é computacionalmente inviável achar um par (x, x') dado que F(x) = F(y). Essas "colisões" referem-se a casos onde entradas de dados diferentes produzem o mesmo resultado.

No presente trabalho, a função hash criptográfica desempenha um papel crucial na segurança e integridade dos dados. As funções hash são utilizadas para criar uma "impressão digital" única de cada bloco de dados, garantindo que qualquer alteração nos dados resulte em um hash completamente diferente. Isso permite a detecção de modificações não autorizadas, essencial para manter a confiabilidade e a imutabilidade da blockchain.

No Hyperledger Fabric, as funções hash são usadas tanto na geração de identificadores únicos para transações quanto na verificação da integridade dos blocos, reforçando a confiança no sistema descentralizado proposto pelo protótipo.

2.1.2 *Blockchain*

Blockchain é, resumidamente, uma tecnologia de livro-razão distribuída e descentralizada, a qual fornece informações que podem ser registradas, mantidas, e compartilhadas por uma comunidade (ANDARGACHEW; GEDEFAW; BIRARA, 2023).

Mell et al. (2018) define *blockchain* como *ledgers* resistentes a manipulação, implementados de maneira distribuída e sem uma autoridade central. Se destaca que um "*ledger*" pode ser compreendido como um livro-razão, atuando como um repositório de estados de objetos e informações acerca do histórico de transações que conduziram aos estados registrados. Dessa maneira, mesmo diante da possibilidade de alteração do estado dos objetos, o histórico permanece imutável (Hyperledger Fabric, 2020).

Rodeck e Curry (2022) relatam que uma *blockchain* é um livro-razão digital distribuído que armazena dados de qualquer tipo, com o aspecto da descentralização tornando a tecnologia única.

A IBM (2023), relata que uma *blockchain* "fornece informações imediatas, compartilhadas e completamente transparentes armazenadas em um livro de registros imutável que pode ser acessado apenas por membros da rede autorizada". A organização ainda destaca a versatilidade da tecnologia, por poder atuar em "pedidos, pagamentos, e muito mais".

A Amazon Web Services (2023), afirma que a tecnologia “permite o compartilhamento transparente de informações na rede de uma empresa. [...] não é possível excluir nem modificar a cadeia sem o consenso da rede.” Além disso, a empresa levanta a possibilidade de criar um livro-razão “inalterável ou imutável” para monitoramento de transações.

É notória a repetição de conceitos como “transparência”, “compartilhamento”, e “imutabilidade”. Então, antes de abordar a estrutura ou até mesmo os componentes de uma *blockchain*, será estabelecido e explicado algumas das bases na qual a tecnologia *blockchain* se pauta.

A descentralização citada nas definições anteriores ocorre em dois sentidos. Primeiro, não há a necessidade de um intermediário “de confiança”, ou autoridade responsável pela autorização de transações. Além disso, a rede, seus dados, e as decisões ficam descentralizadas entre várias máquinas. Assim, caso seja feita uma alteração em um registro de uma máquina, para que isso persista e se consolide na rede, teria que ser uma alteração aceita por todas. Isso reforça a segurança contra a adulteração de dados. Haber e Stornetta (1991) e Nakamoto (2009) relatam:

> A inovação da *blockchain* é que há um *ledger* distribuído, ou seja, existe em muitas cópias interligadas. Esse fato altera profundamente a dinâmica de poder e governança sob as informações, uma vez que descarta-se a necessidade de confiar em uma autoridade central.

No quesito de imutabilidade, as transações não podem ser alteradas ou desfeitas. Caso ocorra algum erro em uma transação, é necessário criar uma nova transação para a correção da transação errada, no entanto, nunca pode se ter um bloco simplesmente alterado.

O conceito de consenso marca a *blockchain* uma vez que para que uma transação seja registrada, é necessário que a maioria dos participantes da rede deem o seu consentimento para que aquela transação efetivamente se torne parte da rede, e seja distribuída entre as máquinas. O conceito de consenso será aprofundado posteriormente neste capítulo.

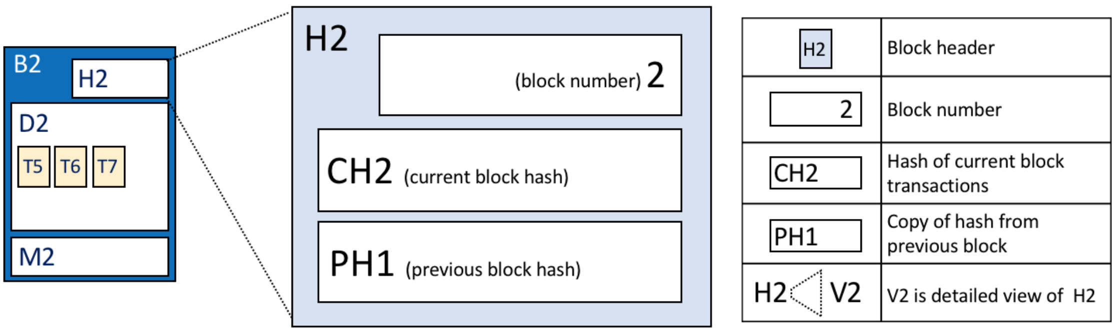
Estabelecidos os conceitos na qual o *blockchain* se pauta, pode-se olhar para a estrutura de um *blockchain*, começando pelo bloco. Como o próprio nome diz, uma *blockchain* tem sua estrutura no formato de uma cadeia de blocos. Essa cadeia, como já foi visto, é distribuída pelas máquinas participantes da rede. Cada máquina é chamada de “nó”. Para facilitar o entendimento, primeiro será analisado o bloco.

O bloco é composto por três seções, que serão descritas com base na documentação do Hyperledger Fabric, uma vez que esta será a framework utilizada ao longo do desenvolvimento do trabalho.

- Número do bloco, que será iniciado em 0 e acrescido de forma sequencial a cada novo bloco anexado à *blockchain*.
- Hash do bloco, referente a todas as transações que o bloco contém.
- Hash do cabeçalho do bloco anterior.
- Dados do bloco: uma lista de transações em ordem.
- Metadados do bloco, contendo informações como a assinatura do criador do bloco para ser verificado pelos nós da rede. Esta seção não é utilizada para o cálculo do hash do bloco, diferente das seções acima. A seção ainda conta com um indicador de validade,

A estrutura acima está ilustrada na figura 3, abaixo.

Figura 3 - Estrutura de um bloco



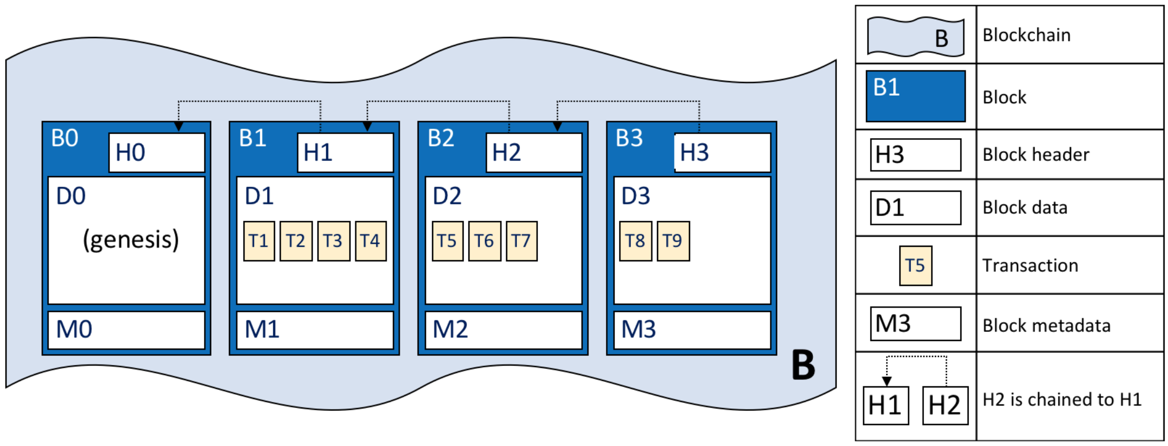
Fonte: Hyperledger Fabric (2020).

Desse modo, é possível começar a visualizar como a cadeia de blocos se forma, e entender um pouco melhor o motivo de *blockchain* ser considerado tão seguro. Cada bloco está conectado ao seu antecessor não apenas em uma sequência, mas também por meio de seu hash. Assim, qualquer tentativa de manipulação em um bloco já integrado à rede resultaria na alteração do hash desse bloco. Isso, por sua vez, influencia o hash no bloco subsequente, indicando uma modificação no bloco anterior. Essa interconexão torna qualquer alteração na cadeia imediatamente detectável.

O bloco de número zero, conhecido como bloco de gênese, é considerado a raiz da *blockchain*. O processo de criação desse bloco difere dos demais, uma vez que incorpora informações sobre a configuração inicial e transações cruciais para a aplicação naquele momento, denominadas transações de gênese (Documentação do Hyperledger Fabric, 2020). Quando um novo membro é adicionado à rede, o bloco de gênese é compartilhado com esse novo integrante, a menos que haja um bloco de reconfiguração disponível.

Pode-se visualizar uma representação de uma *blockchain* abaixo, após o encadeamento de novos blocos.

Figura 4 - Estrutura de uma *blockchain*



Fonte: Hyperledger Fabric (2020)

Existem algumas arquiteturas possíveis para uma *blockchain*, e a escolha de uma arquitetura deve considerar as necessidades da rede.

As redes públicas caracterizam-se pela ausência de permissões, proporcionando acesso livre e global - ou seja, ingresso livre para um novo nó, e acesso irrestrito às atividades da rede. No entanto, enfrentam desafios em termos de escalabilidade, além de maior tempo de validação para novos blocos. As redes públicas aproveitam e reforçam ao máximo os conceitos de auto governança, transparência e descentralização. As arquiteturas públicas são amplamente adotadas, sendo a arquitetura utilizada nos projetos de Bitcoin e Ethereum - possivelmente os dois exemplos mais famosos de aplicação de *blockchain* na atualidade. É comum, nesse tipo de arquitetura, que os nós recebam uma recompensa pelo seu trabalho na rede, logo, é uma arquitetura bastante utilizada em modelos de criptomoedas.

Redes privadas dependem de um intermediário de confiança para determinar as permissões que os diferentes nós terão na rede. Assim, é garantido maior privacidade e rastreabilidade ao modelo, já que todos os nós da rede devem ser conhecidos pela autoridade central, tornando mais fácil a responsabilização de um nó por uma transação inválida.

Blockchains de consórcio são um meio termo entre as duas arquiteturas anteriores - múltiplas organizações irão definir os nós responsáveis pelo consenso na rede, oferecendo um modelo mais descentralizado comparado às redes privadas. No entanto, demanda a confiança entre organizações, além de apresentar outras dificuldades logísticas.

Existem, ainda, as *blockchains* híbridas, que são controladas por uma organização, mas com o nível de liberdade de uma *blockchain* pública, juntando os benefícios de uma autoridade central com a flexibilidade de uma rede mais aberta (WANG; WEGRZYN, 2021).

2.1.3 Contratos inteligentes

Os contratos inteligentes, conhecidos como "smart contracts" em inglês, foram idealizados por Nick Szabo, um pioneiro em criptografia e criptomoedas, em 1994 (CNN BRASIL, 2023). Naquela época, Szabo concebeu um sistema que executaria automaticamente contratos, sem a necessidade de intermediários. No entanto, a implementação dessa ideia era inviável devido às limitações tecnológicas da época (CNN BRASIL, 2023). Atualmente, os contratos inteligentes tornaram-se um componente essencial na arquitetura *blockchain*. Alharby e Moorsel (2017, tradução nossa) definem:

> Um contrato inteligente é um código executável que roda na *blockchain* para facilitar, executar, e reforçar os termos de um acordo. O principal objetivo de um contrato inteligente é automaticamente executar os termos de um acordo quando as condições especificadas forem atingidas. Assim, contratos inteligentes oferecem baixos custos de transação comparado com os sistemas tradicionais, que requerem um terceiro de confiança para reforçar e executar os termos de um acordo.

Portanto, podemos conceber um contrato inteligente, de maneira simplificada, como um algoritmo que é executado ao atingir condições pré-definidas. Dessa forma, essas são os conjuntos de regras armazenados na *blockchain* que determinarão as ações a serem tomadas e em quais circunstâncias.

Existem contratos inteligentes determinísticos e não determinísticos. Um contrato inteligente determinístico, ao ser executado, não depende de nenhuma informação externa ao *blockchain*, já o não-determinístico depende de tal informação - por exemplo, a temperatura atual a ser utilizada no algoritmo (ALHARBY, MOORSEL, 2017), ou um pagamento automático a um fornecedor quando mercadorias chegam ao porto esperado (Amazon Web Services, 2023). A IBM (2023) enfatiza que, embora os contratos possam ser criados por programadores, há uma quantidade crescente de organizações que utilizam *blockchain* para o comércio podem utilizar templates, interfaces web, e outras ferramentas online para simplificar a criação dos contratos.

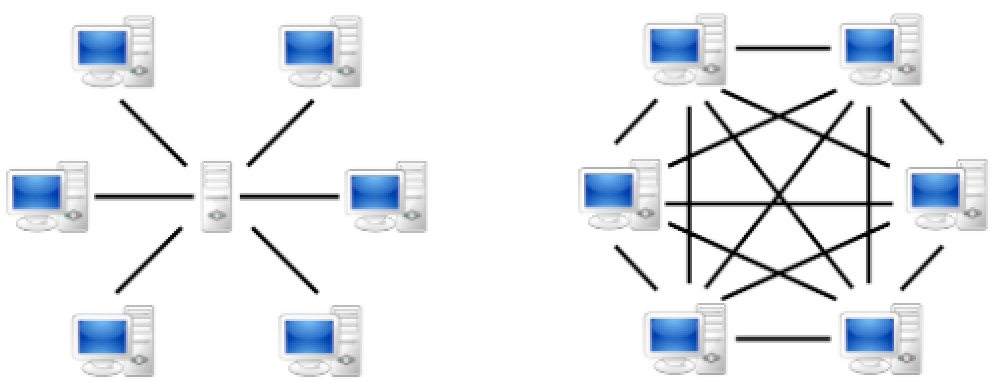
Ao utilizar-se o Hyperledger Fabric para implementar o protótipo de uma blockchain, os contratos inteligentes são fundamentais para automatizar e executar acordos. No contexto do Hyperledger Fabric, esses contratos, chamados de *chaincodes*, permitem a definição de regras de negócio que são imutáveis e transparentes, garantindo que todas as partes envolvidas cumpram suas obrigações sem a necessidade de intermediários. A implementação de contratos inteligentes no protótipo de blockchain facilita a criação de aplicações descentralizadas robustas, aumentando a confiança e a eficiência das transações no sistema.

2.1.4 *Peer-to-peer*

A arquitetura peer-to-peer, comumente conhecida como "P2P", pode ser entendida como uma arquitetura ponto a ponto. Esse nome decorre do seu modo de comunicação, no qual cada máquina interage diretamente com a próxima, comunicando-se com seu vizinho para fazer repasses ou solicitações. Assim, o P2P refere-se a sistemas e aplicações que empregam recursos distribuídos para executar atividades de forma descentralizada (MILOJICIC et al., 2003).

Na arquitetura P2P, os pontos da rede são interdependentes para a obtenção de informações, processar recursos, e encaminhar solicitações, entre outras tarefas. Em outras palavras, a comunicação no sistema é crucial para o sucesso de uma atividade. Esta arquitetura distribuída baseia-se nos conceitos de colaboração, uma vez que cada nó desempenha um papel essencial na troca eficiente de informações e recursos. Quanto maior a quantidade de clientes utilizando a rede, maior a capacidade da mesma, pois maior a quantidade de servidores (BONA, 2022).

Figura 5 - Arquitetura tradicional servidor cliente (esquerda) e arquitetura peer to peer (direita)



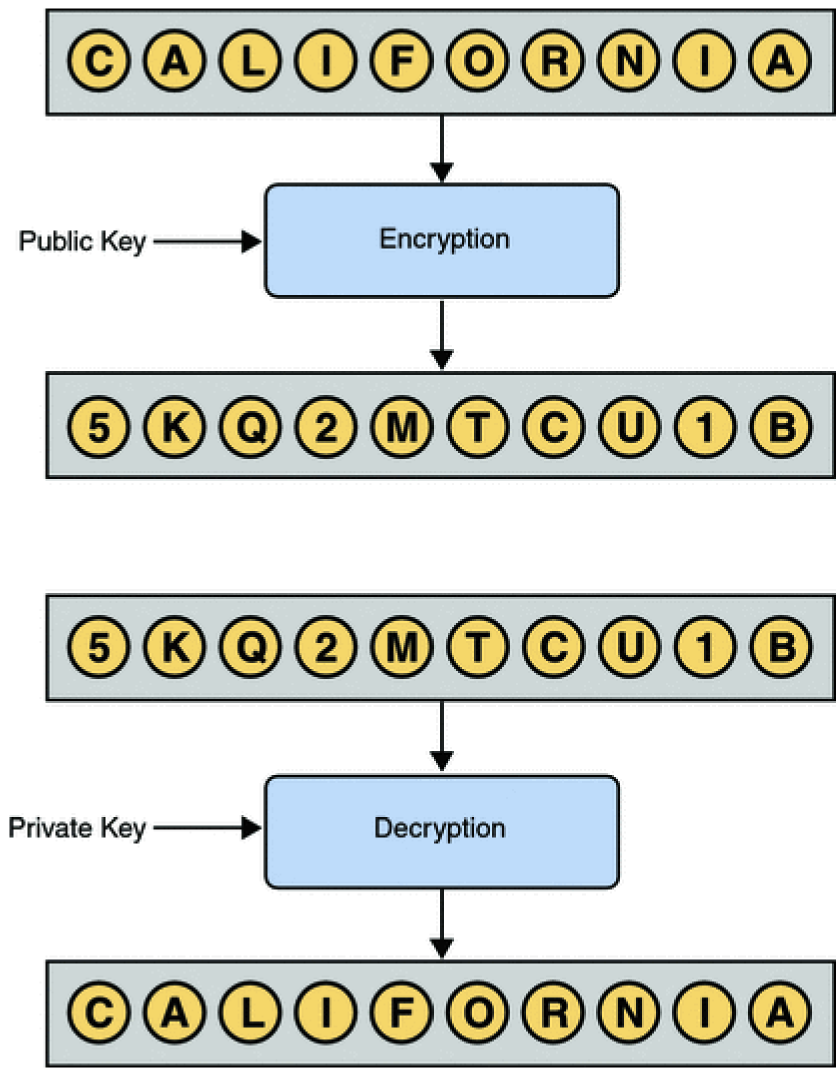
Fonte: Macedo, Moraes, Júlio

É essa a arquitetura na qual o desenvolvimento de uma *blockchain* se baseia, onde cada nó pode atuar tanto como cliente quanto como servidor, resultando assim nas transações descentralizadas. É um componente vital no Hyperledger Fabric, fundamental para a descentralização, resiliência e escalabilidade da rede blockchain.

2.1.5 Criptografia de chave pública

Para garantir a autenticação e identificação de participantes em uma rede *blockchain*, muitas vezes é utilizado a criptografia de chaves assimétricas, também conhecida como criptografia de chave pública. A criptografia de chave pública é baseada no conceito de um par de chaves, matematicamente relacionadas, sendo uma pública, e outra privada, trabalhando juntas, de forma complementar, para criptografar e descriptografar.

Figura 6 - Esquema de criptografia de chave pública



Fonte: Oracle (2010)

Nesse contexto, dados criptografados com uma chave pública só podem ser descriptografados com a chave privada correspondente. Em contrapartida, dados que foram criptografados com uma chave pública só podem ser descriptografados com a chave privada correspondente. A chave pública é disponibilizada para outros participantes, para possibilitar a criptografia dos dados que serão enviados a esse usuário, mas a chave privada é mantida em sigilo, acessível apenas pelo usuário que a gerou (ORACLE, 2010).

Em uma rede utilizando criptografia de chave pública, cada participante possui um par de chaves, conforme explicado anteriormente. Quando um usuário A deseja enviar uma mensagem para um usuário B, com a garantia de que apenas o usuário B consiga lê-la, mesmo se interceptada por um terceiro no caminho, esse sistema é empregado. Basta o usuário A utilizar a chave pública do usuário B para criptografar a mensagem, e enviá-la. Ao receber a mensagem, o usuário B utiliza a sua chave privada, que está disponível só pra ele, para descriptografar a mensagem. Somente ele, com sua chave privada, consegue desfazer a criptografia (IBM 2015, 2022).

Esse modelo demonstra eficácia pois é computacionalmente inviável reverter a criptografia sem a chave privada correspondente, mas é relativamente simples descriptografar a mensagem quando se possui essa chave privada. A segurança desse método reside na dificuldade prática de quebrar a criptografia sem acesso à chave privada apropriada. A facilidade do método é dada uma vez que a chave pública, ao ser distribuída para os participantes, facilita a criptografia por parte dos remetentes.

Essa abordagem oferece um meio seguro de transmitir mensagens sensíveis em ambientes inseguros, garantindo que apenas o destinatário pretendido possa decifrar e compreender o conteúdo da mensagem.

A criptografia de chave pública no Hyperledger Fabric é fundamental em todas as etapas de processamento de transações, desde a assinatura e verificação inicial até a validação e comunicação segura, garantindo a autenticidade, integridade e confidencialidade das informações na rede blockchain.

2.1.6 Protocolos de consenso

Esta seção foi escrita paralelamente ao artigo de 2020, "*Analysis of the main consensus protocolos of blockchain*", por Shijie Zhang e Jong-Hyouk Lee, adaptado para as necessidades deste trabalho.

Em sistemas distribuídos, surge um problema que não está presente em arquiteturas tradicionais. Em arquiteturas distribuídas, como previamente visto, cada nó é simultaneamente um servidor e um cliente, e os nós precisam se comunicar entre si. Diferentemente das arquiteturas centralizadas, onde um servidor único toma as decisões, em sistemas distribuídos, vários servidores estão dispersos. Para assegurar uma tomada de decisão mais robusta e íntegra, é essencial que esses servidores se comuniquem entre si.
Desse modo, surge a definição de consenso: o processo para alcançar acordo sobre o estado válido, dentro de um sistema distribuído. Em outras palavras, a presença de múltiplos nós introduz a possibilidade de falhas individuais. Logo, caso algum nó esteja indisponível ou apresentando comportamento malicioso, isso pode prejudicar o processo de consenso no sistema.

Nesse contexto, a complexidade adicionada pela descentralização demanda estratégias para garantir que, mesmo em caso de falhas ou nós maliciosos, o sistema seja capaz de alcançar consenso de maneira confiável. Para enfrentar esse desafio, sistemas distribuídos, como *blockchains*, adotam abordagens para assegurar a consistência de consenso entre nós. Essas abordagens são chamadas de protocolos de consenso.

Os protocolos de consenso podem ser usados juntos, para se complementar e garantir ainda mais robustez na tomada de decisão frente a nós prejudiciais ao sistema. Nessa seção, será abordado alguns dos protocolos de consenso mais comumente utilizados pelas *blockchains*.

Proof of Work (PoW), ou prova de trabalho, é um protocolo de consenso em que os nós da rede competem entre si para resolver um problema criptográfico complexo. O primeiro nó que encontrar a solução a submete aos outros nós para validação, e, ao ser validada, o nó vencedor garante o direito de adicionar um novo bloco à *blockchain*. É um processo computacionalmente custoso, uma vez que a complexidade do problema a se resolver pelos nós é constantemente ajustado para exigir um esforço significativo no processo de resolução, mas ainda manter um bom ritmo na criação de novos blocos. Uma desvantagem desse protocolo é o alto consumo de energia demandado.

O protocolo de tolerância a Falhas Bizantinas Prática (PBFT) é um protocolo de Tolerância a Falhas Bizantinas bastante prático porém com baixa complexidade. Um cliente solicita uma transação a um nó primário. Este nó propõe a transação para os demais nós, que, por sua vez, indicam sua prontidão para aceitar a proposta na fase de pré-visualização. Após isso, os nós confirmam que estão preparados para comprometer-se com a transação. Então, o nó primário indica que a transação é aceita, e os demais concordam em confirmar as transações. Esse processo, segundo Zhang e Lee, garante menor tolerância a falhas, mas tem um gasto energético considerado dispensável, em comparação (ZHANG; LEE, 2020).

Segundo a documentação do Hyperledger Fabric (HYPERLEDGER, 2020), os protocolos de consenso são componentes críticos no Hyperledger Fabric, garantindo a integridade e a consistência dos dados na rede blockchain. Sua importância se manifesta em várias etapas do funcionamento do Hyperledger Fabric. Inicialmente, quando uma transação é proposta por um cliente, ela deve ser endossada por um conjunto de *peers* conforme as políticas de endosso definidas para a rede. Esses *peers* executam a transação e produzem assinaturas digitais (*endorsements*) que confirmam a execução correta da transação segundo as regras do *chaincode*.

Após a obtenção dos endossos necessários, a transação é enviada ao serviço de ordenação, que é responsável por sequenciar todas as transações de forma global. Aqui, entra em ação o protocolo de consenso, que determina a ordem exata das transações antes de serem agrupadas em blocos. O Hyperledger Fabric suporta diferentes mecanismos de consenso, como o protocolo Raft, que é amplamente utilizado por sua simplicidade e eficiência, garantindo que todos os *nodes* da rede concordem sobre a ordem das transações.

Finalmente, os blocos ordenados são distribuídos aos *peers* da rede, que verificam as transações contidas no bloco quanto à conformidade com as políticas de endosso e a integridade geral. Uma vez verificadas, as transações são então aplicadas ao ledger local de cada peer, garantindo que todos mantenham uma cópia consistente e atualizada do ledger.

Em resumo, os protocolos de consenso no Hyperledger Fabric são vitais em todas as etapas do processamento de transações, desde a coleta de endossos e a ordenação global das transações até a verificação final e a atualização do ledger. Eles asseguram que a rede funcione de maneira harmoniosa e que todos os participantes tenham uma visão consistente e confiável dos dados, mantendo a integridade e a confiabilidade da blockchain.

2.1.6 Hyperledger Fabric

Estabelecidos os conceitos chave referentes a *blockchain* e seu funcionamento, pode-se iniciar o entendimento da plataforma que será utilizada para o desenvolvimento do trabalho proposto, a Hyperledger Fabric. Sobre a própria plataforma, o Hyperledger Fabric (2020) relata:

> A Linux Foundation fundou o projeto Hyperledger em 2015 para promover as tecnologias de *blockchain* entre indústrias. [...] A Hyperledger Fabric é um dos projetos de *blockchain* do Hyperledger. Como outras tecnologias de *blockchain*, ela

> possui uma livro-razão, usa contratos inteligentes e é um sistema pelo qual os participantes gerenciam suas transações.

A plataforma apresenta-se como uma "solução *blockchain* corporativa abrangente, mas personalizável", privada, e permissionada. A plataforma oferece todos os principais componentes importantes para uma *blockchain*: o livro-razão compartilhado, contratos inteligentes, privacidade, e protocolos de consenso, com diversas configurações possíveis para cada um desses ítens. A plataforma oferece suporte a linguagens como Node e Java para o desenvolvimento de *chaincodes*, proporcionando acessibilidade e flexibilidade aos desenvolvedores. Logo, o Hyperledger Fabric promete adaptabilidade e simplicidade para o desenvolvimento de uma *blockchain*, adotando uma arquitetura modular para garantir maior flexibilidade para as necessidades de cada aplicação.

Assim, apresenta-se, então, o modelo Hyperledger Fabric, e seus principais recursos e conceitos, de acordo com a documentação do Hyperledger Fabric (Hyperledger Fabric, 2020):

- Ativos (*assets*): os ativos, podem abranger desde elementos tangíveis aos intangíveis. Possibilita a representação de uma variedade de ativos digitais, podem, por exemplo, representar os dados que serão enviados.
- Pares *(peers)*: os nós no Hyperledger Fabric são denominados pares. Esses pares constituem os participantes da rede que mantêm cópias compartilhadas do livro-razão e colaboram no processo de consenso.
- Chaincode: os contratos inteligentes no Hyperledger Fabric são denominados *chaincodes*, representando a lógica de negócio na plataforma. Esses *chaincodes* têm a capacidade de manipular ativos e são implantados em canais específicos.
- Canais: a implementação de canais no Hyperledger Fabric cria sub-redes privadas dentro da *blockchain* principal. Esses canais possibilitam o isolamento de informações e transações entre participantes específicos, fornecendo uma camada adicional de privacidade e segregação de dados.
- Livro-razão (*ledger*): o livro-razão, ou *ledger*, no Hyperledger Fabric, é imutável, composto e sequenciado por meio de uma cadeia de blocos. Cada canal na rede possui seu próprio livro-razão, e cada par mantém uma cópia do livro-razão para cada canal em que participa. Essa estrutura descentralizada contribui para a integridade e transparência das transações na rede.
- Ordenadores (*orderers)*: um nó de ordenação ou um "ordenador" ordena as transações, compondo o serviço de ordenação, que organiza a sequência e realiza o "empacotamento" dos blocos.

Atualmente, o Hyperledger Fabric já é utilizado na blockchain criada no projeto Jornada do Estudante, e continuará sendo utilizado para realizar a implementação proposta no presente trabalho.

2.1.3 MiniFabric

Estabelecida a importância do Hyperledger Fabric no processo de criação de uma blockchain, ainda entende-se que a ferramenta pode ser bastante complexa por apresentar alguns desafios no seu uso. Assim, a HyperLedger Foundation, entidade responsável pelo Hyperledger Fabric, disponibiliza o MiniFabric como uma ferramenta de "início rápido" para o Hyperledger Fabric, para facilitar o aprendizado, a administração, e o desenvolvimento de uma rede Fabric.

O MiniFabric foi projetado para descomplicar as interações com o Hyperledger Fabric, tornando mais acessível a configuração e gerenciamento de redes Fabric, uma vez que elimina a necessidade de interagir diretamente com aspectos mais complexos da plataforma. Com apenas alguns comandos rápidos, torna-se bastante simples atualizar uma chaincode, invocar transações, e até mesmo subir uma rede Fabric.

Assim, será utilizado o MiniFabric para a criação e gestão da rede Fabric no trabalho em questão, buscando aproveitar a facilidade que a ferramenta oferece através da sua abstração.

2.1.4 Docker

O Docker, conforme o site oficial da organização, "ajuda os desenvolvedores a compilar, compartilhar, rodar, e verificar aplicações em qualquer lugar, sem configuração de ambiente ou gerenciamento" (DOCKER, 2024). A ferramenta utiliza o conceito de containers e imagens para encapsular software juntamente com suas dependências necessárias para execução. Ao executar uma imagem, é criado um novo container, com as configurações especificadas, na máquina, apresentando consistência de forma independente do sistema operacional do hospedeiro e das suas configurações locais.

A ferramenta destaca-se por oferecer a opção de portabilidade a aplicações de complexidade elevada, tornando irrelevante as diferenças entre sistemas operacionais ou requisitos específicos da aplicação, por garantir as dependências necessárias através do container.

No presente trabalho, o Docker desempenha um papel fundamental, uma vez que a ferramenta encapsula os componentes da rede Hyperledger Fabric, em contêineres individuais, simplificando a manutenção dos componentes, além de garantir a presença das suas dependências.

2.1.5 Go

Go é uma linguagem de programação criada pela Google, lançada em código livre, que tem ganhado destaque pela sua eficiência, facilidade de aprender, e desempenho em sistemas concorrentes. Algumas características como a coleta de lixo, o suporte à concorrência, e a compilação rápida tornam Go uma boa escolha para o desenvolvimento de aplicações escaláveis de alto desempenho.

No contexto de blockchain, Go pode ser utilizada em plataformas como o Hyperledger Fabric para o desenvolvimento de *chaincodes*, uma vez que oferece capacidade para lidar com operações simultâneas de forma eficaz. Além disso, por ter uma sintaxe clara e tipagem forte, facilita a criação de um código seguro e menos suscetível a erros. Desse modo, será a linguagem utilizada para desenvolver as chaincodes propostas no presente trabalho.

2.2 ENEM E PROCESSOS SELETIVOS

Nesta segunda parte, serão esclarecidos assuntos referentes à prova do ENEM e dos processos que ocorrem de forma dependente dele, necessários para entender o impacto do projeto proposto, além do escopo do trabalho.

2.2.1 Exame Nacional do Ensino Médio (ENEM)

Sobre o ENEM, o Ministério da Educação (2023) afirma:

> O Exame Nacional do Ensino Médio (ENEM) foi instituído em 1998, com o objetivo de avaliar o desempenho escolar dos estudantes ao término da educação básica. Em 2009, o exame aperfeiçoou sua metodologia e passou a ser utilizado como mecanismo de acesso à educação superior.

A prova ocorre ao longo de dois dias, com uma pausa de uma semana entre o primeiro e segundo dia, e é constituída por 180 questões, divididas em quatro categorias, conhecidas popularmente como “áreas de conhecimento” - Ciências Humanas e suas Tecnologias, Ciências da Natureza e suas Tecnologias, Linguagens, Códigos, e Suas Tecnologias, e

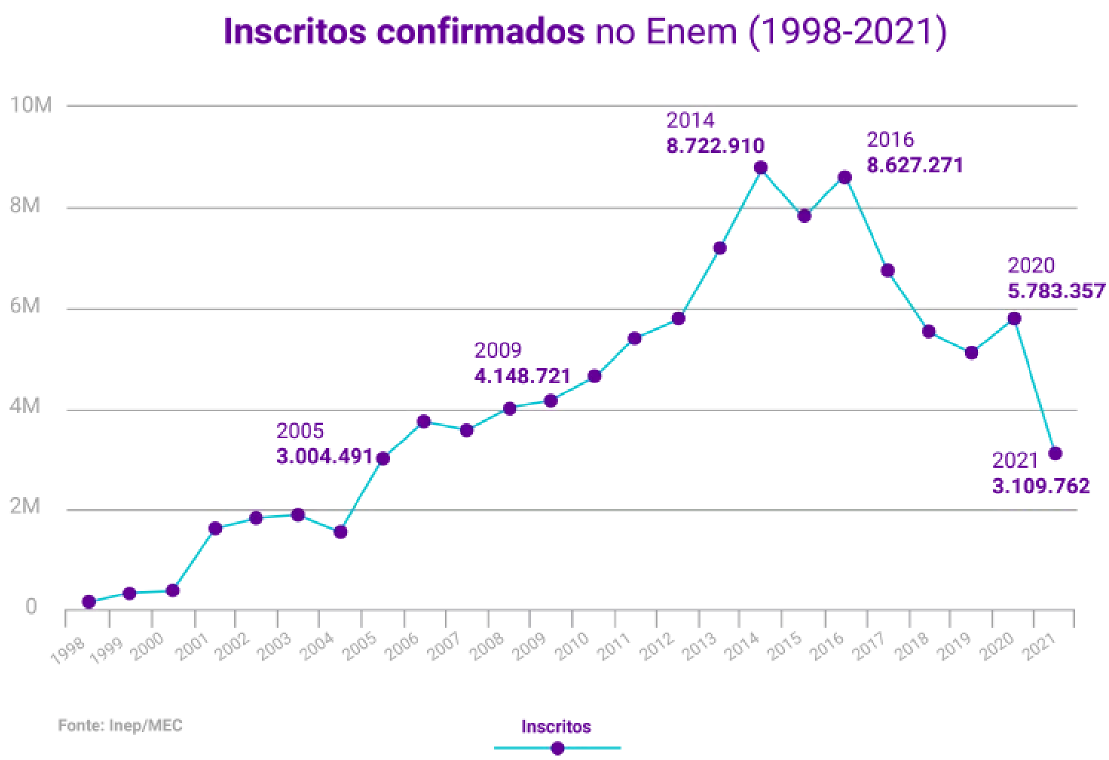
Matemática e suas Tecnologias. Além disso, a prova conta com uma redação (CAMPOS, 2018).

Embora a prova tenha crescido por conta de alguns ajustes feitos nos seus primeiros anos, como a concessão de isenção de taxa de inscrição aos estudantes de escolas públicas, ou como a ampliação de horas para a realização da prova, existem alguns marcos consideráveis que contribuíram para o crescimento da prova, crescimento que pode ser conferido abaixo, na figura 7.

Em 2005 foi criado o ProUni, processo que será abordado posteriormente nessa seção, resultando em um considerável aumento no número de inscritos, principalmente daqueles que já haviam concluído o ensino médio (CAMPOS, 2018). Em 2009, com a criação do SiSU, as matrizes de referência da prova são reformuladas com base nas matrizes do Exame Nacional para Certificação de Competências de Jovens e Adultos (Encceja). Assim, o ENEM ganhou o formato que hoje é aplicado em todo o território nacional. Em 2010, a prova tornou-se pré-requisito para o Fies - processo que também será abordado posteriormente - e teve seu sistema de inscrição aprimorado. Em 2013, a prova passou a ser adotada por grande parte das instituições federais como critério de seleção. Em 2015, foi criada a categoria de “treineiro” para o exame, identificando candidatos que fazem a prova com o intuito de obter uma autoavaliação (Inep, 2018).

Hoje, com 25 anos de história, o ENEM é conhecido por muitos como a maior prova do Brasil (BATISTA, 2018).

Figura 7 - Inscritos confirmados no Enem (1998 - 2021)



Fonte: Fundação Telefônica Vivo, 2021.

2.2.2 Sistema de Seleção Unificada (SiSU)

O Sistema de Seleção Unificada, popularmente conhecido como SiSU, é uma plataforma acessada exclusivamente pela internet que reúne vagas ofertadas por instituições públicas de ensino superior do país todo, tanto federais, quanto estaduais. O processo abre duas vezes por ano, e os candidatos podem concorrer a vagas abertas em qualquer lugar do país, independente de onde moram ou onde foi prestado a prova do ENEM (MEC, 2023). Sobre o SiSU, em conjunto com o ENEM, Castro (2023) afirma:

> [...] ele unificou os processos seletivos das universidades federais e teve adesão também para algumas estaduais, facilitando a disputa por vagas e possibilitando que os candidatos pudessem analisar de maneira mais objetiva o leque de opções disponíveis e qual a nota necessária para ingressar em cada curso.

Ao inscrever-se no SiSU, o sistema recupera todas as notas do estudante, do ENEM válido para o processo seletivo em questão, e o candidato poderá escolher até duas opções de curso em cada processo seletivo. Ao longo do processo, o estudante poderá modificar suas opções, conforme seu interesse. No painel do candidato, é possível ver sua colocação na modalidade escolhida, além da nota de corte das opções. Candidatos que não foram selecionados inicialmente poderão optar pela lista de espera das modalidades em que ele estava inscrito. Ainda segundo o Ministério da Educação, as vagas seguem a distribuição da Lei de Cotas, e podem contar com pesos diferentes por área de conhecimento, conforme a instituição achar interessante para cada curso ofertado.

Para garantir a possibilidade de se inscrever em uma vaga através do SiSU, o candidato, segundo o MEC, deve ter realizado a edição do ENEM mais recente antes do processo seletivo o qual deseja concorrer através do SiSU que não obtenha nota zero na redação, e não seja um treineiro.

2.2.3 Programa Universidade Para Todos (Prouni)

O Programa Universidade Para Todos foi criado em 2004, e oferece, para alunos de baixa renda, bolsa de estudos em instituições privadas de ensino superior. O valor da bolsa pode ser equivalente a 50% ou 100% da mensalidade do curso de graduação, desde que o candidato tenha renda familiar bruta mensal de 1,5 salário mínimo até 3 salários mínimos, a depender do valor pretendido da bolsa. Além disso, o candidato deve atender a pelo menos uma condição referente a sua instituição de ensino durante o seu ensino médio, ser pessoa com deficiência, ou ser professor da rede pública, para os cursos de licenciatura e pedagogia. Não é permitido fazer uso da bolsa do Prouni paralelamente a cursar uma instituição pública de ensino superior gratuita - assim, o candidato deverá optar, caso selecionado em ambos os processos, pelo Prouni ou SiSU (Ministério da Educação, 2023).

Segundo a jornalista Beatriz Araujo, em 2023, foram ofertadas 288 mil bolsas pelo programa. Candidatos que receberam nota 0 na redação e não tenham atingido, no mínimo 450 pontos em cada área de conhecimento não podem participar do processo seletivo (ARAUJO, 2023).

2.2.4 Fundo de Financiamento Estudantil (Fies)

O Fies, programa do MEC, criado em 2001, oferece financiamento variável de acordo com a renda familiar, para estudantes que têm uma renda familiar mensal bruta, por pessoa, de até 3 salários mínimos. Caso o estudante tenha renda familiar de até 1,5 salários mínimos, ele pode ter direito a juros zero (ARAUJO, 2023). O estudante deve respeitar os mesmos requisitos de nota mínima que aqueles inscritos no Prouni, ou seja, é vedada a participação daqueles que atingiram nota 0 na redação e obtiveram menos de 450 pontos em cada área de conhecimento.

O Fies conta com 3 etapas: a fase de utilização, carência, e amortização. na fase de utilização, o estudante pagará, a cada três meses, uma taxa de, no máximo, R$ 150,00, referente ao pagamento de juros incidentes sobre o financiamento. Após a formatura, o estudante terá 18 meses de "pausa" para reorganizar seu orçamento, pagando o mesmo valor na mesma frequência que na fase anterior. Na amortização, o saldo devedor será dividido em até 12 anos.

2.2.5 Acesso Único

O Acesso Único, também conhecido como Portal Único de Acesso ao Ensino Superior, foi criado durante o desenvolvimento do Projeto Acesso Único. O projeto do MEC está "alicerçado em princípios como: segurança digital, modernidade, transparência, participação do usuário, integração, inteligência, monitoramento e economicidade" (Ministério da Educação, 2023). O MEC afirma que a plataforma foi desenvolvida "a partir de experiências de estudantes" que se inscreveram em edições passadas dos processos seletivos que são contemplados pelo portal.

Qualquer candidato que busca participar de mais de um processo seletivo, precisa preencher suas informações cadastrais uma única vez, armazenando os dados do candidato desde o primeiro registro, para que o mesmo não precise repetir tais dados ao se inscrever em outros processos seletivos. É necessário apenas que o candidato complemente as informações com base naquilo demandado pelo processo específico pretendido. Os processos contemplados pelo Acesso Único são todos os processos listados acima, ou seja: SiSU, Prouni, e Fies.

3. EXECUÇÃO

Esta seção visa ilustrar como foi o processo de desenvolvimento do protótipo proposto. Primeiro, será explicado como foi o processo de entendimento e coleta dos dados, bem como a definição daqueles que foram escolhidos para participar do protótipo. Além disso, serão explicados os objetivos que a *chaincode* desenvolvida visa atingir. Após isso, será explicado como foi o processo de instalação da rede, os comandos utilizados, e o processo de instalação da *chaincode* desenvolvida, além da própria *chaincode*.

3.1 DEFINIÇÃO DOS REQUISITOS E DADOS

Para iniciar o desenvolvimento, foi feito um levantamento dos dados contemplados ao longo do processo de inscrição do ENEM. A pesquisa e identificação dos dados envolvidos no ENEM não foi uma tarefa fácil, uma vez que não é disponibilizado pelo INEP os dados que serão solicitados em cada edição, durante o processo de inscrição. Assim, só é possível a identificação desses dados quando é feita, de fato, a inscrição.

No intuito de conseguir compreender da melhor forma possível quais dados são envolvidos no processo, algumas estratégias diferentes foram utilizadas.

Primeiramente, foi feita uma simulação de pedido de solicitação de isenção de taxa, para o ano de 2024, na prova do ENEM. Assim, ao simular a solicitação de isenção, pôde-se coletar e registrar quais dados são solicitados ao candidato naquela etapa. No entanto, considerando o calendário de inscrição do ENEM no ano de 2024, não foi possível realizar a simulação de inscrição em si, uma vez que as inscrições abrem em uma data posterior ao momento de desenvolvimento do presente trabalho.

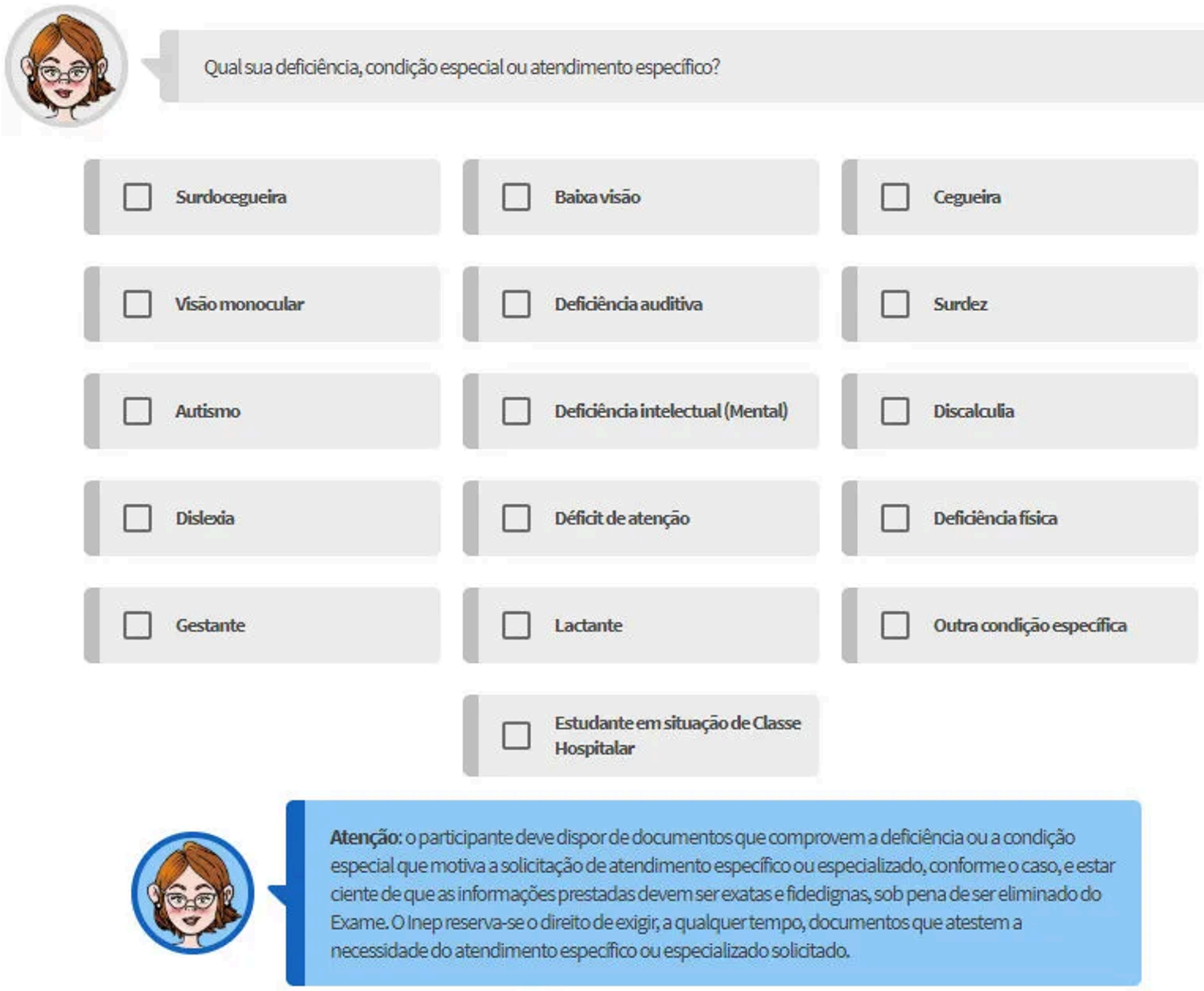
Para complementar a simulação feita, foi feita uma pesquisa extensiva buscando os dados solicitados em processos passados. Assim, foi possível entender quais dados adicionais seriam solicitados aos candidatos, no momento da inscrição na prova, embora nem sempre seja possível entender o formato exato, suas obrigatoriedades, ou possíveis opções de resposta. No entanto, a ideia é familiarizar-se com os dados que são envolvidos, visando, posteriormente, fazer o filtro destes dados, para decidir o que estará presente na blockchain num primeiro momento.

Os dados identificados ao longo da pesquisa estão listados abaixo.

- Documentos:
    - número de Cadastro de Pessoa Física (CPF);
    - número de outro documento de identidade.

- Dados pessoais do candidato:
  - nome completo do pai;
  - nome completo da mãe;
  - sexo;
  - cor/raça;
  - estado civil;
  - nacionalidade;
  - Unidade Federativa (UF) e município de nascimento;
  - endereço.
- Local de prova:
  - UF;
  - município.
- Língua estrangeira:
  - inglês ou espanhol - a opção, por padrão, é inglês.
- Atendimento especializado ou específico:
  - Caso o candidato precisa de algum recurso de acessibilidade, deve informar a deficiência, condição ou atendimento específico, dentre as opções a seguir; Destaca-se, neste quesito, que o candidato deve dispor de documentos que comprovem a informação, mas não foi possível identificar como/se o envio desse documento é solicitado, e, se sim, quando é o processo de envio desse documento. As opções possíveis para solicitação de atendimento especializado podem ser conferidas na figura 8.

Figura 8 - Preenchimento de motivo de necessidade de atendimento específico na inscrição do ENEM



Fonte: INEP (2024)

Ao realizar sua inscrição no ENEM, ou realizar sua solicitação de isenção de taxa, também deve ser preenchido um questionário socioeconômico, de forma obrigatória. As perguntas contidas no questionário estão listadas abaixo.

- Questionário socioeconômico:
  - Até que série seu pai, ou o homem responsável por você, estudou?
  - Até que série sua mãe, ou a mulher responsável por você, estudou?
  - A partir da apresentação de algumas ocupações divididas em grupos ordenados, indique o grupo que contempla a ocupação mais próxima da ocupação do seu pai ou do homem responsável por você. (Se ele estiver trabalhando, escolha uma ocupação pensando no último trabalho dele).
  - A partir da apresentação de algumas ocupações divididas em grupos ordenados, indique o grupo que contempla a ocupação mais próxima da

ocupação da sua mãe ou da mulher responsável por você. (Se ela não estiver trabalhando, escolha uma ocupação pensando no último trabalho dela).
- Incluindo você, quantas pessoas moram atualmente na sua casa?
- Você possui renda?
- Qual é a renda mensal de sua família? (Considere a sua renda com a das pessoas que moram com você.)
- Em sua casa, você ou a pessoa responsável pela sua família costuma contratar os serviços de um(a) empregado(a) doméstico(a)?
- Em sua casa, existe banheiro?
- Em sua casa, existe quarto para dormir?
- Incluindo você, as pessoas com quem você mora têm carro?
- Incluindo você, as pessoas com quem você mora têm motocicleta?
- Em sua casa, existe geladeira?
- Em sua casa, existe freezer independente (vertical ou horizontal)? (Não considere a segunda porta da geladeira.)
- Em sua casa, existe máquina de lavar roupa? (Não considere o tanquinho.)
- Em sua casa, existe micro-ondas?
- Em sua casa, existe aspirador de pó?
- Em sua casa, existe aparelho de TV?
- Em sua casa, existe TV por assinatura?
- Em sua casa, existe acesso à internet por rede wi-fi?
- Em sua casa, existe computador/notebook?
- Incluindo você, as pessoas com quem você mora têm telefone celular?
- Em que tipo de escola você frequentou ou frequenta o Ensino Médio?

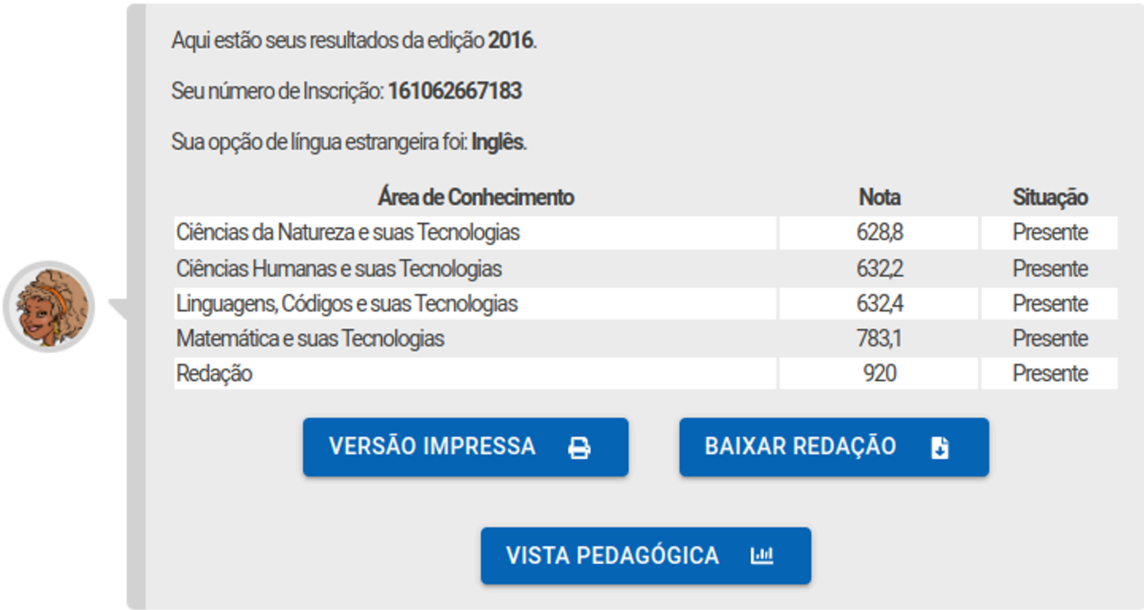
Estabelecidos os dados que são fornecidos no período anterior à prova, a próxima pesquisa que foi realizada buscou entender quais dados o candidato recebe após realizar a sua prova - ou seja, aquilo que faz parte do resultado do candidato, e aquilo que é incluído na sua devolutiva. Para gerar esse entendimento, foi feito uma retrospectiva com a devolutiva passada da própria autora.

Assim, os dados e insumos recebidos pelo candidato estão listados a seguir.

- Dados presentes no resultado do candidato, conforme pode-se observar na figura 8
  - ano da edição;

- número de inscrição;
- nome completo;
- CPF;
- língua estrangeira que optou realizar a prova, no momento de inscrição;
- nota e situação por área de conhecimento, logo, temos:
    - Nota na Área de Conhecimento de Ciências da Natureza e suas Tecnologias;
    - Situação na Área de Conhecimento de Ciências da Natureza e suas Tecnologias (Presente/Ausente);
    - Nota na Área de Conhecimento de Ciências Humanas e suas Tecnologias;
    - Situação na Área de Conhecimento de Ciências Humanas e suas Tecnologias (Presente/Ausente);
    - Nota na Área de Conhecimento de Linguagens, Códigos, e suas Tecnologias;
    - Situação na Área de Conhecimento de Linguagens, Códigos e suas Tecnologias (Presente/Ausente);
    - Nota na Área de Conhecimento de Matemática e suas Tecnologias;
    - Situação na Área de Conhecimento de Matemática e suas Tecnologias (Presente/Ausente);
    - Nota na Área de Conhecimento de Redação;
    - Situação na Área de Conhecimento de Redação (Presente/Ausente).

Figura 9 - Resultado do ENEM



Aqui estão seus resultados da edição **2016**.

Seu número de Inscrição: **161062667183**

Sua opção de língua estrangeira foi: **Inglês**.

Área de Conhecimento	Nota	Situação
Ciências da Natureza e suas Tecnologias	628,8	Presente
Ciências Humanas e suas Tecnologias	632,2	Presente
Linguagens, Códigos e suas Tecnologias	632,4	Presente
Matemática e suas Tecnologias	783,1	Presente
Redação	920	Presente

Fonte: INEP (2024).

Além dos dados listados acima, o candidato ainda pode optar por baixar sua redação daquela edição, ou analisar a vista pedagógica da redação daquela edição. Caso ele opte por baixar sua redação, ele poderá visualizar uma cópia escaneada da sua folha de redação daquela edição.

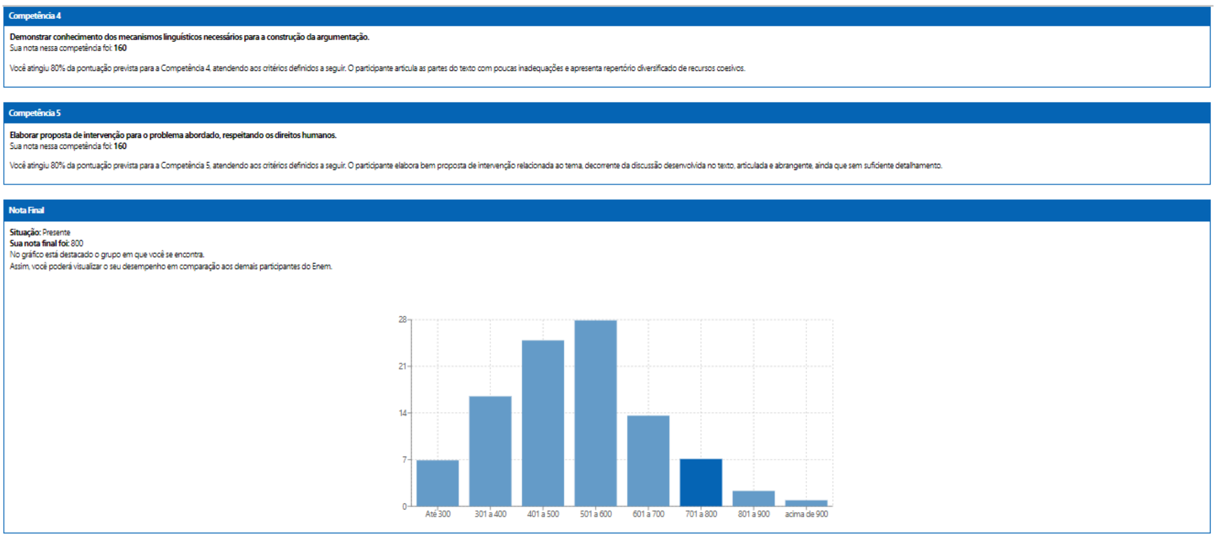
Na vista pedagógica, é fornecido:

- Dados pessoais:
    - número de inscrição;
    - nome;
    - CPF.
- Nota por competência analisada:
    - Nota da Competência 1;
    - Nota da Competência 2;
    - Nota da Competência 3;
    - Nota da Competência 4;
    - Nota da Competência 5.

A nota é acompanhada por uma porcentagem do quanto foi atingido, considerando a nota máxima possível por competência, juntamente com uma breve descrição da competência, e um resumo do desempenho do candidato, de acordo com a faixa de nota que o mesmo atingiu para uma competência qualquer. O candidato também pode visualizar, em um gráfico fornecido, como desempenhou comparado com os outros candidatos naquele ano, onde há uma análise frequência de notas, e em qual faixa o candidato se encaixa, de acordo com sua nota final, naquele ano.

Um exemplo da vista pedagógica, contendo os dados mencionados acima, pode ser conferido na figura 10, abaixo.

Figura 10 - Vista Pedagógica da redação do ENEM



Fonte: INEP (2024)

Após a análise dos dados fornecidos no momento de inscrição do candidato, e dos dados fornecidos no resultado do candidato, definiu-se quais são considerados importantes para participarem do protótipo, num primeiro momento. Assim, os dados selecionados estão listados abaixo.

- Dados fornecidos pelo candidato, que foram selecionados para o protótipo:
  - Documentos:
    - número de Cadastro de Pessoa Física (CPF);
    - número de outro documento de identidade.
  - Dados pessoais do candidato:
    - nome completo do pai;
    - nome completo da mãe;
    - sexo;

        - cor/raça;
        - estado civil;
        - nacionalidade;
        - Unidade Federativa (UF) e município de nascimento;
        - endereço.
    - Local de prova:
        - UF;
        - município.
    - Língua estrangeira:
        - inglês ou espanhol.
- Dados fornecidos no resultado, que foram selecionados para o protótipo:
    - ano da edição;
    - número de inscrição;
    - nome completo;
    - CPF;
    - língua estrangeira que optou realizar a prova, no momento de inscrição;
    - nota e situação por área de conhecimento, logo, tem-se:
        - Nota na Área de Conhecimento de Ciências da Natureza e suas Tecnologias;
        - Situação na Área de Conhecimento de Ciências da Natureza e suas Tecnologias (Presente/Ausente);
        - Nota na Área de Conhecimento de Ciências Humanas e suas Tecnologias;
        - Situação na Área de Conhecimento de Ciências Humanas e suas Tecnologias (Presente/Ausente);
        - Nota na Área de Conhecimento de Linguagens, Códigos, e suas Tecnologias;
        - Situação na Área de Conhecimento de Linguagens, Códigos e suas Tecnologias (Presente/Ausente);
        - Nota na Área de Conhecimento de Matemática e suas Tecnologias;
        - Situação na Área de Conhecimento de Matemática e suas Tecnologias (Presente/Ausente);
        - Nota na Área de Conhecimento de Redação;
        - Situação na Área de Conhecimento de Redação (Presente/Ausente).

Enxerga-se grande potencial de ganho ao incluir os dados do questionário socioeconômico, uma vez que possibilitaria um rastreio das informações fornecidas, para análises futuras socioeconômicas a serem realizadas pelo INEP. Caso incluído, em um momento posterior, poderia ser analisado juntamente com a solicitação da isenção de taxa e seu resultado, para garantir maior compreensão sobre a diferença entre a realidade dos alunos que solicitam a isenção, e aqueles que não solicitam, além de ser um ponto interessante para rastreio, uma vez que poderia vir a economizar tempo para as IES num momento futuro. Um exemplo de utilidade dessa informação poderia ser a aprovação imediata de bolsas para alunos de baixa renda, caso o aluno já tenha recebido essa isenção previamente, uma vez que já comprovou-se sua necessidade. Caso um aluno busque se inscrever em uma cota racial que ele não sinalizou ter no seu momento de inscrição do ENEM, isso já poderia ser identificado, uma vez que existiria rastreabilidade sobre esses fatos. No entanto, entende-se que isso foge do escopo do presente trabalho, logo, o questionário socioeconômico não fará parte dos dados do protótipo neste momento.

O mesmo pode ser dito sobre as informações contidas na vista pedagógica da redação. Entende-se que são informações que podem ser valiosas em um momento futuro de análises e pesquisas para melhor compreender a nota de redação no geral e suas tendências, mas não foi incluído nesse momento, por entender que há menor prioridade do que os dados escolhidos. Os gráficos de desempenho da redação não foram considerados relevantes para a inclusão.

O objetivo é projetar um contrato - referente ao candidato inscrito, e referente ao resultado do ENEM. A parte do contrato referente ao candidato inscrito salvará sua inscrição, juntamente com seus dados relevantes. Assim, deve ser possível que o contrato que tratará da inscrição registre a inscrição na blockchain, e a localize, trazendo os dados referentes àquela inscrição, localizada através do número de inscrição do candidato inscrito. A parte do contrato inteligente referente ao resultado do candidato deve desempenhar essa mesma função, ou seja, registrar o resultado na blockchain, e o localizar, com base no número de inscrição.

Estabelecidos os dados e aquilo que é considerado importante estar presente na blockchain, foi feita a instalação do ambiente.

3.2 INSTALAÇÃO DO AMBIENTE

Uma vez que um dos objetivos do trabalho é que a *chaincode* esteja presente na arquitetura do projeto atual do jornada, foi feito a instalação do ambiente do jornada, para que o trabalho seja desenvolvido na própria rede.

Após fazer a instalação do código-fonte do Jornada, as variáveis de ambiente referentes ao Minifabric foram configuradas. Isso garante o funcionamento correto do Minifabric na pasta em que a rede será instalada - neste caso, será instalado em uma pasta chamada de “mywork”, seguindo o padrão da documentação do Minifabric, além do padrão do laboratório.

Feita a configuração do ambiente, pôde-se iniciar a instalação. Para realizar a instalação, o projeto já conta com um *script* que faz todo o processo de forma automatizada. Assim, bastou executar o arquivo de instalação. Esse *script* inicializa a rede base e as *chaincodes* do projeto, as applications, APIs, e os *containers* do Docker. Assim, concluída a execução do script, a rede está pronta para ser utilizada.

3.3 IMPORTAÇÕES, FUNÇÕES BÁSICAS, E DEFINIÇÃO DE *ASSETS*

Para iniciar o desenvolvimento das *chaincodes*, primeiro foram definidos os ativos, ou *assets*, que seriam utilizados, e o formato de entrada de dados. Dois assets foram definidos, um referente à inscrição do candidato, para a *chaincode* de Inscrição, chamado de Inscricao, e outro, referente ao resultado para uma aplicação em uma edição qualquer, chamado de ResultEnem. Nessas estruturas, foram definidos os atributos de acordo com as informações que foram consideradas essenciais para o protótipo, de acordo com aquilo definido na seção 4.1. Pode-se perceber que, na estrutura Inscricao, que representa as inscrições, foi adicionado um atributo de número de inscrição, o NumInscricao. Isso ocorre pois, conforme visto anteriormente, o resultado contém esse atributo como identificador único. Logo, entende-se que esse identificador é gerado no momento da inscrição do candidato. Na rede, o NumInscricao será de importância máxima, uma vez que será a chave de busca para os objetos - tanto o de Inscricao, quanto o de Result.

O formato de dados escolhido foi de *JavaScript Object Notation* (JSON), por ser amplamente utilizado, legível, e por familiaridade com o JSON por trabalhos anteriores. Assim, o JSON fornece um formato simples para realizar a entrada de dados na blockchain.
Os assets podem ser conferidos abaixo, já estruturados com seus atributos e o formato de entrada.

```
type Inscricao struct {
   NumInscricao              string
`json:"numInscricao"`
   CPF                       string    `json:"CPF"`
   DataNascimento            time.Time
`json:"dataNascimento"`
   Nome                      string    `json:"nome"`
   NomePai                   string    `json:"nomePai"`
   NomeMae                   string    `json:"nomeMae"`
   Sexo                      string    `json:"sexo"`
   Cor                       string    `json:"cor"`
   RG                        RG        `json:"RG"`
   EstadoCivil               string    `json:"estadoCivil"`
   Nacionalidade             string
`json:"nacionalidade"`
   Naturalidade              Municipio
`json:"naturalidade"`
   Endereco                  string    `json:"endereco"`
   AnoEdicao                 string    `json:"anoEdicao"`
   LinguaEstrangeira         string
`json:"linguaEstrangeira"`
   Escolaridade              string
`json:"nivelEscolaridade"`
   TipoEnsinoMedio           string
`json:"tipoEnsinoMedio"`
   InstituicaoEnsinoMedio    string
`json:"instituicaoEnsinoMedio"`
   OpcaoAtendimentoEspecial string
`json:"opcaoAtendimentoEspecial"`
   LocalProvaUF              string
`json:"localProvaUF"`
   LocalProvaMunicipio       string
`json:"localProvaMunicipio"`
}

type Municipio struct {
   CodigoMunicipio           string
`json:"codigoMunicipio"`
   NomeMunicipio             string `json:"nomeMunicipio"`
   UF                        string `json:"UF"`
   EhEstrangeiro             bool   `json:"ehEstrangeiro"`
   NomeMunicipioEstrangeiro string
`json:"nomeMunicipioEstrangeiro"`
}

type Result struct {
   NumInscricao              string  `json:"numInscricao"`
   CPF                       string  `json:"cpf"`
   AnoEdicao                 string  `json:"anoEdicao"`
   LinguaEstrangeira         string
`json:"linguaEstrangeira"`
   NotaCienciasNatureza      float64
`json:"notaCienciasNatureza"`
   SituacaoCienciasNatureza string
`json:"situacaoCienciasNatureza"`
   NotaCienciasHumanas       float64
`json:"notaCienciasHumanas"`
   SituacaoCienciasHumanas  string
`json:"situacaoCienciasHumanas"`
   NotaLinguagens            float64
`json:"notaLinguagens"`
```

```
   SituacaoLinguagens       string
`json:"situacaoLinguagens"`
   NotaMatematica           float64
`json:"notaMatematica"`
   SituacaoMatematica       string
`json:"situacaoMatematica"`
   NotaRedacao              float64 `json:"notaRedacao"`
   SituacaoRedacao          string
`json:"situacaoRedacao"`
}

type RG struct {
   Numero         string `json:"numero"`
   OrgaoExpedidor string `json:"orgaoExpedidor"`
   UF             string `json:"UF"`
}
```

Após a estruturação dos *assets*, foi feito um estudo daquilo que seria necessário com base naquilo contido em cada *asset*, além daquilo definido como objetivos do protótipo, conforme também definido na seção 4.1. Assim, primeiro foi iniciado o desenvolvimento das funções de inscrição. Primeiro, é definido o *package* em que a *chaincode* irá rodar, além de fazer a importação de pacotes necessários.

```
package main

import (
   "encoding/json"
   "fmt"
   "time"

   customErrors "chaincodeErrors"

   "github.com/hyperledger/fabric-chaincode-go/shim"
   "github.com/hyperledger/fabric-protos-go/peer"
)
```

No trecho *import()*, é importado, de acordo com a documentação dos pacotes envolvidos:

- “encoding/json”: o pacote próprio do Go para codificação e decodificação de JSON. Esse pacote dará funções como “*marshal*” e “*unmarshal*”, os quais farão, respectivamente, a codificação e decodificação do JSON;
- “fmt”: o pacote implementa operações de entrada e saída formatadas, fornecendo ferramentas para mostrar dados na tela, e receber dados do usuário de forma organizadas e formatada;

- "*time*": com o pacote "*time*", funções de medição e formatação de tempo serão disponibilizadas. Isso será utilizado para trabalhar com datas;
- "https://github.com/hyperledger/fabric-chaincode-go/shim": o pacote oferece acesso a APIs para a *chaincode* acessar suas variáveis de estado, contexto de transação, e chamar outras *chaincodes*. Resumidamente, o pacote fornece um meio para que o código possa interagir com o estado da *blockchain*, além de outras *chaincodes*;
- "https://github.com/hyperledger/fabric-chaincode-go/peer": o pacote *"peer"* fornece APIs para que a *chaincode* interaja com o ambiente do nó *peer*, possibilitando o acesso a informação sobre as transações que ocorrem na rede, permite acesso ao livro razão, e facilita a comunicação com outros *peers*.

Por fim, cria-se o tipo "Enem".

```
type Enem struct {
}
```

A criação dessa estrutura ocorre por conta do funcionamento do Go, para que as funções sejam endereçadas à estrutura "Enem". Como a linguagem não trabalha com o conceito de classes, a estrutura é criada desse modo. Assim, garante-se que a permissão de chamada das funções endereçadas à esse tipo podem ser feitas apenas pelo tipo "Enem". Como tem-se várias funções com o mesmo nome em um projeto blockchain - como Init, e Invoke, que serão explicados mais adiante, essa tipagem faz a diferenciação entre as funções.
Após as importações e criação de estruturas, iniciou-se o desenvolvimento das funções. Manteve-se o padrão do projeto de organização, tanto de ordem alfabética dos assets, quanto a ordem que as funções ficam organizadas no código. A primeira função é a função Init.

```
func (e *Enem) Init(stub shim.ChaincodeStubInterface)
peer.Response {
   return shim.Success(nil)
}
```

A função "*Init*" é chamada durante a instância ou atualização do contrato na rede. Não contém lógica específica, apenas retornando uma resposta de sucesso, para indicar que a inicialização foi realizada.

```
func main() {
   err := shim.Start(new(Enem))
   if err != nil {
       fmt.Printf("Erro iniciando a chaincode: %s", err)
   }
}
```

A função main é o ponto de entrada de qualquer aplicação em Go. Assim, quando o programa é executado, a execução começa a partir da função acima. é chamado o shim.*Start*(*new*(Enem)), que tem a responsabilidade de iniciar a *chaincode* no Hyperledger Fabric, preparando a mesma para começar a receber e processar transações. Assim, é no bloco de código mostrado acima que se cria uma nova instância da *chaincode* Enem.

3.4 *CHAINCODE* DESENVOLVIDA

Para facilitar o entendimento dos próximos blocos de código, o fluxo seguido será o fluxo natural e intuitivo de um candidato. Primeiro, deve ser feita a inscrição, e, após feita, o mesmo pode ter acesso às informações contidas nela. Após a inscrição, deve ser registrado o resultado, para visualização posterior. Assim, tem-se quatro funções principais, além de uma função que chama essas quatro, chamada de *Invoke*. Todas as funções pertencem a uma estrutura "Enem", conforme explicado anteriormente. As quatro funções abordadas serão: *storeInscricao*(), *readInscricaoByNumInscricao*(), *storeResultado*(), e, finalmente, *readResultadoByNumInscricao*(). Para aproximar-se de um contexto mais realista, as seguintes regras de negócio foram definidas:

- não deve ser possível cadastrar mais de uma inscrição com o mesmo número de inscrição;
- não deve ser possível cadastrar mais de um resultado para o mesmo número de inscrição;
- não deve ser possível cadastrar um resultado sem que haja uma inscrição com o mesmo número de inscrição, uma vez que não se pode ter um resultado sem que se tenha uma inscrição.

Definidas as regras, pode-se iniciar o entendimento do código fonte.

A primeira função abordada será aquela que representa o ato de registrar uma inscrição na rede, representada pela função *storeInscricao*(). A função recebe dois parâmetros, mas apenas um é passado de forma externa, chamado de args - uma redução da palavra *arguments*, em inglês. "args" é um objeto JSON, que será processado para armazenar as informações referentes àquela inscrição na rede. O outro parâmetro, o primeiro, chamado de *stub*, é uma interface que permite a interação com o *ledger*. A função inicia realizando o *unmarshall* do JSON, para converter o objeto na estrutura Inscricao. Caso tenha algum erro nessa transformação, será obtido um erro e a função irá parar sua execução. Após essa transformação, tem-se já a inscrição, representada pela variável inscricao, do tipo Inscricao. A seguir, é definido uma chave para salvar e buscar o objeto na rede. A chave, no caso das funções do objeto de Inscricao, é uma junção da expressão INSCRICAO_ com o número de inscrição presente na inscrição sendo tratada naquele momento. Após a formação da chave, é feita uma validação para garantir que já não há uma inscrição feita com aquele mesmo número de inscrição. Se houver, ou caso algum imprevisto ocorra durante o processo, será lançado um erro, e a execução da função será interrompida. Caso contrário, a inscrição novamente é convertida para um JSON, e será armazenada no *ledger*, com sua chave única que foi criada. Por fim, caso todos os passos sejam concluídos com sucesso, retorna nulo, indicando a ausência de erros no processo. Existe também a chamada de uma função que imprime, nos *logs* dos *peers*, uma mensagem referente ao sucesso do processo de armazenamento da inscrição.

```
func (e *Enem) storeInscricao(stub
shim.ChaincodeStubInterface, args string) error {
   const funcName = "StoreInscricao"

   var inscricao Inscricao
   err := json.Unmarshal([]byte(args), &inscricao)
   if err != nil {
       errorMsg :=
customErrors.NewMarshallingError(funcName, args, err)
       return errorMsg
   }

   inscricaoKey := "INSCRICAO_" + inscricao.NumInscricao

   inscricaoExistente, err := stub.GetState(inscricaoKey)
   if err != nil {
       errorMsg := customErrors.NewGenericError(funcName,
"Erro ao verificar a            existencia do numero de
inscricao", err)
       return errorMsg
```

```
    }
    if inscricaoExistente != nil {
        errorMsg := customErrors.NewGenericError(funcName,
"Numero de inscricao ja tem inscricao atrelada", nil)
        return errorMsg
    }

    inscricaoJSON, err := json.Marshal(inscricao)
    if err != nil {
        errorMsg :=
customErrors.NewMarshallingError(funcName, args, err)
        return errorMsg
    }

    err = stub.PutState(inscricaoKey, inscricaoJSON)
    if err != nil {
        errorMsg :=
customErrors.NewUpdateWorldStateError(funcName, err)
        return errorMsg
    }
    fmt.Println("Inscricao armazenada, sob o numero de
inscricao ", inscricao.NumInscricao)
    return nil
}
```

Seguindo o fluxo lógico do ENEM, a próxima função é a função que fornece a visualização da inscrição e as informações atreladas à inscrição. Essa função é a *readInscricaoByNumInscricao*(). A função recebe o stub, assim como a anterior, e um número de inscrição. A função segue a mesma lógica de montar a chave de busca, como é feito na função de armazenar uma inscrição, que é usada para buscar o registro no *ledger*, através da chamada do stub.*GetState*(). Nessa busca, retorna-se um JSON e um erro. Caso o erro não seja nulo, retorna-se o erro. Se o JSON for nulo, é informado que não há inscrição com o número, seguido pelo número de inscrição fornecido. Caso exista a inscrição, ela é convertida para seu devido formato, e retornado, para visualização.

```
func (e *Enem) readInscricaoByNumInscricao(stub
shim.ChaincodeStubInterface, numInscricao string)
(*Inscricao, error) {
    const funcName = "ReadInscricaoByNumInscricao"
    inscricaoKey := "INSCRICAO_" + numInscricao

    inscricaoEnemJSON, err := stub.GetState(inscricaoKey)
```

```
    if err != nil {
        errorMsg := customErrors.NewGenericError(funcName,
numInscricao, err)
        fmt.Println("Erro ao obter dados sob o numero de
inscricao ", numInscricao, "-", err)
        return nil, errorMsg
    }
    if inscricaoEnemJSON == nil {
        fmt.Println("Não ha inscricao com o numero ",
numInscricao)
        return nil, nil
    }

    var inscricao Inscricao
    err = json.Unmarshal(inscricaoEnemJSON, &inscricao)
    if err != nil {
        errorMsg :=
customErrors.NewMarshallingError(funcName, "Inscricao",
err)
        return nil, errorMsg
    }
    return &inscricao, nil
}
```

Assim, conclui-se a explicação referente às funções da estrutura de Inscrição. A seguir, serão abordadas as funções da estrutura de Resultado.

Conforme explicado, as funções em si têm o mesmo objetivo que as funções já abordadas, com algumas validações ou detalhes que mudam, para as necessidades do contexto. A função de armazenamento, *storeResultado*(), recebe o *stub* e o args, assim como na função de *storeInscricao*(). O args é convertido na estrutura de resultado. em seguida, é montada a chave de busca e armazenamento para os resultados no ledger. A chave é formada pela expressão RESULTADO_, seguido pelo número de inscrição. São feitas validações para garantir que já há inscrição registrada com o número de inscrição contido no resultado, conforme definido nas regras de negócio. Além disso, é feita uma validação para garantir que não existe resultado já cadastrado com aquele mesmo número de inscrição. Se todas as validações passarem, é usada a chave definida para realizar o armazenamento do resultado no *ledger*. Assim como na função de armazenamento de inscrição, caso não tenha nenhum erro ao longo da função, não é retornado nada por ela, indicando a ausência de falha no processo.

```
func (e *Enem) storeResultado(stub
shim.ChaincodeStubInterface, args string) error {
    const funcName = "StoreResultado"

    var result Result
    err := json.Unmarshal([]byte(args), &result)
    if err != nil {
        errorMsg :=
customErrors.NewMarshallingError(funcName, args, nil)
```

```
        return errorMsg
    }

    resultadoKey := "RESULTADO_" + result.NumInscricao

    inscricaoExistente, err :=
e.readInscricaoByNumInscricao(stub, result.NumInscricao)
    if err != nil {
        errorMsg := customErrors.NewGenericError(funcName,
"Erro ao verificar a existencia da inscricao", err)
        return errorMsg
    }
    if inscricaoExistente == nil {
        errorMsg := customErrors.NewGenericError(funcName,
"Nao ha registro de inscricao atrelado ao numero de
inscricao fornecido, logo, nao pode ser cadastrado um
resultado", nil)
        return errorMsg
    }
    existingResultadoBytes, err :=
stub.GetState(resultadoKey)
    if err != nil {
        errorMsg := customErrors.NewGenericError(funcName,
"Erro ao verificar a existencia do resultado", err)
        return errorMsg
    }
    if existingResultadoBytes != nil {
        errorMsg := customErrors.NewGenericError(funcName,
"Ja existe resultado atrelado ao numero de inscricao
fornecido", nil)
        return errorMsg
    }
    resultJSON, err := json.Marshal(result)
    if err != nil {
        errorMsg :=
customErrors.NewMarshallingError(funcName, args, nil)
        return errorMsg
    }
    err = stub.PutState(resultadoKey, resultJSON)
    if err != nil {
        errorMsg :=
customErrors.NewUpdateWorldStateError(funcName, nil)
        return errorMsg
    }
    return nil
}
```

Além da função de armazenamento do resultado, tem-se a função que permite a sua visualização. A *readResultadoByNumInscricao*() funciona de forma análoga à função *readInscricaoByNumInscricao*(), apenas realizando os devidos ajustes na chave de busca, tal como na função *storeResultado*, além das mensagens de erro. Caso não surja um erro ao longo do processo, é retornado o resultado atrelado ao número de inscrição que a função recebeu como parâmetro, com todas as suas devidas informações.

```
func (e *Enem) readResultadoByNumInscricao(stub
shim.ChaincodeStubInterface, numInscricao string)
(*Result, error) {
   const funcName = "ReadResultadoByNumInscricao"

   resultadoKey := "RESULTADO_" + numInscricao

   resultJSON, err := stub.GetState(resultadoKey)
   if err != nil {
       errorMsg := customErrors.NewGenericError(funcName,
numInscricao, err)
       fmt.Println("Erro ao obter dados com o numero de
inscricao ", numInscricao, "-", err)
       return nil, errorMsg
   }
   if resultJSON == nil {
       fmt.Println("Não ha resultado com o numero de
inscricao ", numInscricao)
       return nil, nil
   }
   var result Result
   err = json.Unmarshal(resultJSON, &result)
   if err != nil {
       errorMsg :=
customErrors.NewMarshallingError(funcName, "Result", err)
       return nil, errorMsg
   }
   fmt.Println("Sucesso ao obter o resultado sob o numero
de inscricao ", result.NumInscricao)
   return &result, nil
}
```

Por fim, o método *Invoke*() é o que efetivamente faz a chamada das quatro funções principais citadas até agora. A função recebe como parâmetro a interface para interagir com o ledger, e a função e parâmetro recebidos por ele - representados pelas variáveis fn e args, respectivamente. É no *Invoke*() que algumas validações são feitas, como garantir que, nas funções será recebido apenas um parâmetro. Além disso, a função retorna um peer.*Response*, indicando o resultado da invocação - nas funções de *store*, por exemplo, caso o objeto seja salvo na rede, é retornado o shim.*Success*, que apontará o *status* 200, indicando sucesso na operação após sua realização. Caso tenha algum problema, o retorno padrão o indicará com um erro. A função complementa e completa as informações retornadas pelas funções as quais o *Invoke*() chama.

```
func (e *Enem) Invoke(stub shim.ChaincodeStubInterface)
peer.Response {

   fn, args := stub.GetFunctionAndParameters()
```

```
   switch fn {

   case "StoreInscricao":
       if len(args) != 1 {
           errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args), nil)
           return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       err := e.storeInscricao(stub, args[0])
       if err != nil {
          if chainError, ok :=
err.(customErrors.ChaincodeError); ok {
               return
customErrors.ToPeerResponse(chainError.Info())
           }
           return shim.Error(err.Error())
       }
       return shim.Success(nil)

   case "ReadInscricaoByNumInscricao":
       if len(args) != 1 {
           errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args), nil)
           return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       inscricao, err :=
e.readInscricaoByNumInscricao(stub, args[0])
       if err != nil {
           return shim.Success(nil)
       }
       if inscricao == nil {
           errorMsg :=
customErrors.NewGenericError("ReadInscricaoByNumInscricao"
, "Nao ha registro de inscricao atrelado o numero de
inscricao fornecido", nil)
           return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       inscricaoInBytes, err := json.Marshal(inscricao)
       if err != nil {
           errorMsg :=
customErrors.NewInvalidStructError(fn, "Inscricao", err)
           return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       return shim.Success(inscricaoInBytes)

   case "StoreResultado":
       if len(args) != 1 {
           errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args), nil)
           return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       var result Result
       err := json.Unmarshal([]byte(args[0]), &result)
       if err != nil {
           errorMsg :=
customErrors.NewMarshallingError(fn, args[0], err)
```

```
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        err = e.storeResultado(stub, args[0])
        if err != nil {
            if chainError, ok :=
err.(customErrors.ChaincodeError); ok {
                return
customErrors.ToPeerResponse(chainError.Info())
            }
            errorMsg :=
customErrors.NewGenericError("StoreResultado", "Falha ao
armazenar o resultado", err)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        return shim.Success(nil)

    case "ReadResultadoByNumInscricao":
        if len(args) != 1 {
            errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args), nil)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        result, err := e.readResultadoByNumInscricao(stub,
args[0])
        if err != nil {
            return shim.Success(nil)
        }
        if result == nil {
            errorMsg :=
customErrors.NewGenericError("ReadResultadoByNumInscricao"
, "Nao ha registro de resultado atrelado ao numero de
inscricao fornecido", nil)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        resultInBytes, err := json.Marshal(result)
        if err != nil {
            errorMsg :=
customErrors.NewInvalidStructError(fn, "Resultado", nil)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        return shim.Success(resultInBytes)

    default:
        errorMsg :=
customErrors.NewUndefinedFunctionError(fn, nil)
        return customErrors.ToPeerResponse(errorMsg.Info())
    }
}
```

3.5 CONSIDERAÇÕES SOBRE O DESENVOLVIMENTO

Embora as funções em si sejam relativamente simples, no processo de desenvolvimento, dificuldades significativas foram encontradas. Além do desafio de implementar algo bastante diferente do cotidiano, fazê-lo em uma linguagem nova acarretou em dificuldades para entender e ler o código em si.

Durante o desenvolvimento, a utilização do MiniFabric trouxe consigo alguns desafios significativos. Uma dificuldade encontrada foi a falta de clareza nas mensagens de erro, as quais frequentemente não aparentavam oferecer indicativos suficientes para resolver com maior facilidade as causas dos problemas ocasionados. Assim, exigiu-se um esforço considerável no processo de depuração e resolução de problemas. As dificuldades nesse processo intensificaram-se também pelo modo como o MiniFabric e o Jornada do Estudante funcionam ao realizar alterações no código. Para que uma alteração seja efetivamente feita, não basta apenas realizar uma mudança no código fonte. Deve ser executado o comando de *update* do MiniFabric, do modo abaixo, passando como parâmetro a *chaincode* que será atualizada, bem como uma nova versão. Caso a versão fornecida como parâmetro já exista, não será feita a atualização. Assim, o tempo que é despendido em testar uma alteração aumenta, além de tornar o processo mais trabalhoso.

```
python3 installer_dev.py update enem -v 1.1
```

Uma dificuldade bastante peculiar foi encontrada no modo como a blockchain realiza suas buscas. Inicialmente, as funções de leitura dos objetos utilizavam o número de inscrição do candidato como a chave para a busca. Ao realizar os testes, percebeu-se que, ao seguir o fluxo natural de inscrição, na primeira vez, as funções funcionam normalmente. Porém, caso fosse realizada uma segunda leitura, os objetos eram retornados corrompidos, de certa forma - campos da inscrição eram mostrados no resultado, por exemplo. Assim, após uma pesquisa extensa, foi possível entender que isso ocorreu pois a chave das duas buscas era a mesma, e, assim, ocorria algum equívoco ao realizar a busca. Assim, a solução encontrada foi fazer com que a chave de busca fosse precedida pelo objeto o qual se está buscando, para que exista essa diferenciação clara. Após a implementação dessa chave, o erro não foi mais encontrado

4. TESTES

Nesta seção, serão abordados os *outputs* e a execução em si das funções desenvolvidas. Para a execução das funções, será utilizado o terminal da máquina. Primeiro, deve-se navegar até a pasta que contém o MiniFabric. Por padrão, essa pasta se chama “mywork”. É nela que serão executados os comandos que chamarão as funções, passando os parâmetros necessários. Além da mywork, tem-se também a pasta “jornada”, que contém todo o código fonte do projeto, como o instalador, conforme descrito na seção 4.2. É aqui que será desenvolvido o código. Após cada atualização no código fonte, deve-se realizar um comando de *update*, para atualizar a versão do código utilizada pelo MiniFabric, para que as mudanças feitas no código sejam refletidas nos resultados.

Após gerar a versão correta do código, pode-se iniciar o teste do código fonte. Esta seção será dividida em duas partes. Na primeira subseção, será feito o caminho ideal, onde todas as funções devem desempenhar sem erros. Na segunda subseção, alguns erros serão forçados, visando garantir que as validações necessárias serão bem sucedidas.

4.1 CADASTRO DA JORNADA DO CANDIDATO

Com o intuito de fazer a demonstração do jeito mais claro possível, será feito de forma a simular a jornada do candidato ao longo do processo do ENEM. Assim, inicia-se pela inscrição, para depois olhar para o resultado do mesmo. Para realizar essa demonstração, dois objetos JSON foram criados, para representar uma inscrição e um resultado, ambos fictícios. Para aproximar a demonstração da vida real, toda a demonstração utilizará o mesmo número de inscrição, indicando que o processo está fazendo o acompanhamento da mesma inscrição/candidato ao longo do processo.

Assim, inicia-se o registro da inscrição. Para isso, será utilizado a função storeInscricao(). O comando utilizado pode ser conferido abaixo. As chamadas das funções seguem o mesmo padrão. Iniciam-se com o ./minifab, para indicar o uso do MiniFabric, seguido pela palavra-chave invoke, que executa a função na chaincode. Utilizando o -p, indicam-se os parâmetros necessários para realizar a invocação. Para todas as funções criadas neste trabalho, deve-se ter sempre dois parâmetros, entre aspas simples: o nome da função, entre aspas duplas, e o parâmetro que aquela função recebe, no formato de *string*. Logo, para realizar a chamada da função storeInscricao(), o comando fica no formato que pode ser conferido abaixo.

```
./minifab invoke -p '"StoreInscricao",
"{\"NumInscricao\":\"numInc1\",\"CPF\":\"123.456.7
89-00\",\"dataNascimento\":\"1990-05-20T00:00:00Z\
",\"nome\":\"João da Silva\",\"nomePai\":\"Carlos
da Silva\",\"nomeMae\":\"Maria da
Silva\",\"sexo\":\"Masculino\",\"cor\":\"Pardo\",\
"RG\":{\"numero\":\"MG-12.345.678\",\"orgaoExpedid
or\":\"SSP\",\"UF\":\"MG\"},\"estadoCivil\":\"Solt
eiro\",\"nacionalidade\":\"Brasileiro\",\"naturali
dade\":{\"codigoMunicipio\":\"3106200\",\"nomeMuni
cipio\":\"Belo
```

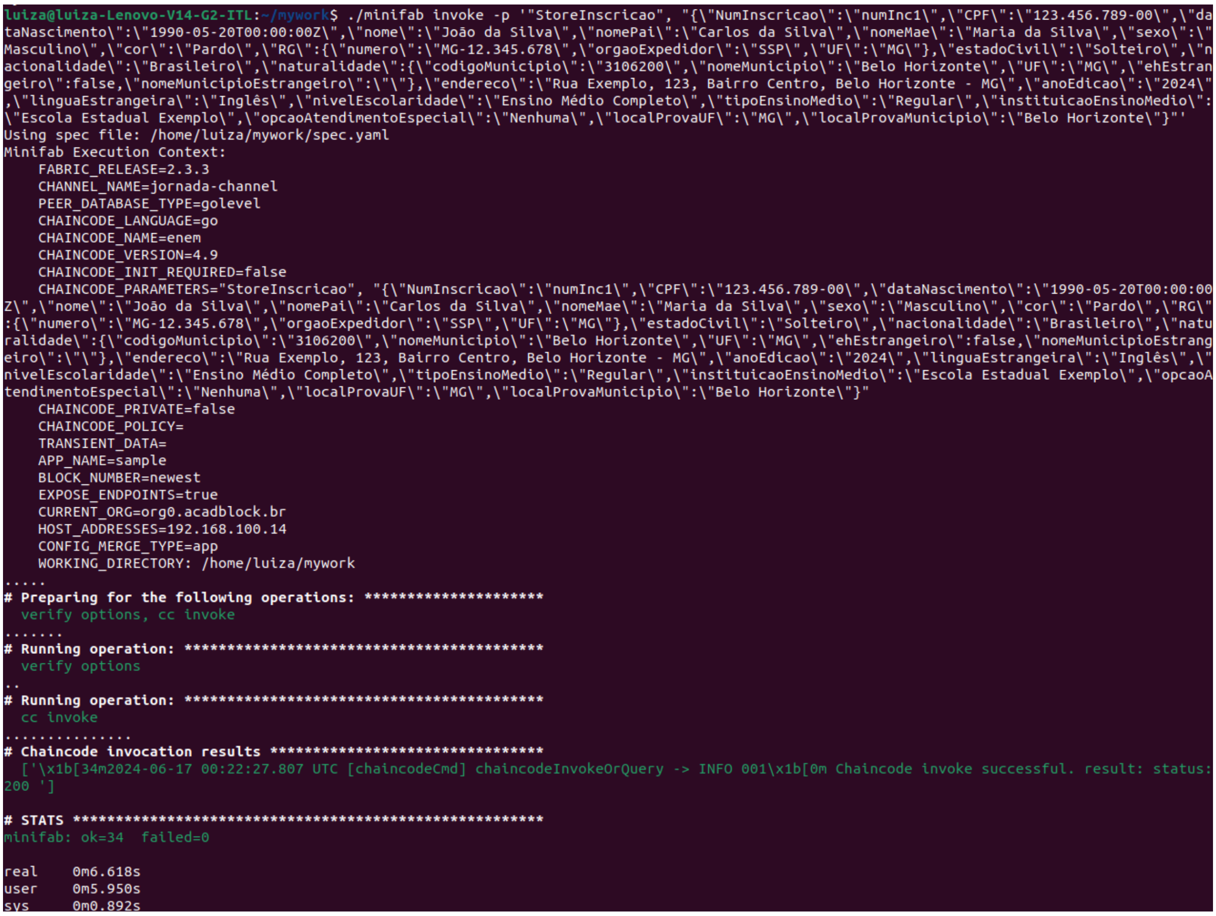
```
Horizonte\",\"UF\":\"MG\",\"ehEstrangeiro\":false,
\"nomeMunicipioEstrangeiro\":\"\"},\"endereco\":\"
Rua Exemplo, 123, Bairro Centro, Belo Horizonte -
MG\",\"anoEdicao\":\"2024\",\"linguaEstrangeira\":
\"Inglês\",\"nivelEscolaridade\":\"Ensino    Médio
Completo\",\"tipoEnsinoMedio\":\"Regular\",\"insti
tuicaoEnsinoMedio\":\"Escola              Estadual
Exemplo\",\"opcaoAtendimentoEspecial\":\"Nenhuma\"
,\"localProvaUF\":\"MG\",\"localProvaMunicipio\":\
"Belo Horizonte\"}"'
```

O segundo parâmetro passado à função, nesse caso, é o JSON, no formato de *string* aceito pelo terminal. Assim, o segundo parâmetro representa o objeto JSON abaixo. As informações fornecidas são fictícias, apenas para intuito de teste do código desenvolvido. Destaca-se o atributo de NumInscricao, que, conforme explicitado previamente, será utilizado como chave para todas as operações feitas ao longo do teste. Assim, pretende-se realizar uma inscrição que utiliza o número de inscrição “numInc1”. É importante ressaltar mais uma vez que esse número deve ser único, sendo o principal identificador de uma inscrição para um candidato em uma edição específica. Ou seja, caso esse candidato realize outra inscrição em outra edição da prova, o número de inscrição seria diferente toda vez, no sistema do ENEM.

```json
{
  "NumInscricao": "numInc1",
  "CPF": "123.456.789-00",
  "dataNascimento": "1990-05-20T00:00:00Z",
  "nome": "João da Silva",
  "nomePai": "Carlos da Silva",
  "nomeMae": "Maria da Silva",
  "sexo": "Masculino",
  "cor": "Pardo",
  "RG": {
    "numero": "MG-12.345.678",
    "orgaoExpedidor": "SSP",
    "UF": "MG"
  },
  "estadoCivil": "Solteiro",
  "nacionalidade": "Brasileiro",
  "naturalidade": {
    "codigoMunicipio": "3106200",
    "nomeMunicipio": "Belo Horizonte",
    "UF": "MG",
    "ehEstrangeiro": false,
    "nomeMunicipioEstrangeiro": ""
  }
```

Ao executar o comando, recebemos a seguinte mensagem, conforme pode ser visualizado na figura 11: “*Chaincode invoke succesful. result: status 200”*. Logo, a função foi executada com sucesso, e a inscrição foi devidamente armazenada na rede.

Figura 11 - Captura de tela da operação *storeInscricao()*



```
luiza@luiza-Lenovo-V14-G2-ITL:~/mywork$ ./minifab invoke -p '"StoreInscricao", "{\"NumInscricao\":\"numInc1\",\"CPF\":\"123.456.789-00\",\"da
taNascimento\":\"1990-05-20T00:00:00Z\",\"nome\":\"João da Silva\",\"nomePai\":\"Carlos da Silva\",\"nomeMae\":\"Maria da Silva\",\"sexo\":\"
Masculino\",\"cor\":\"Pardo\",\"RG\":{\"numero\":\"MG-12.345.678\",\"orgaoExpedidor\":\"SSP\",\"UF\":\"MG\"},\"estadoCivil\":\"Solteiro\",\"n
acionalidade\":\"Brasileiro\",\"naturalidade\":{\"codigoMunicipio\":\"3106200\",\"nomeMunicipio\":\"Belo Horizonte\",\"UF\":\"MG\",\"ehEstran
geiro\":false,\"nomeMunicipioEstrangeiro\":\"\"},\"endereco\":\"Rua Exemplo, 123, Bairro Centro, Belo Horizonte - MG\",\"anoEdicao\":\"2024\"
,\"linguaEstrangeira\":\"Inglês\",\"nivelEscolaridade\":\"Ensino Médio Completo\",\"tipoEnsinoMedio\":\"Regular\",\"instituicaoEnsinoMedio\":
\"Escola Estadual Exemplo\",\"opcaoAtendimentoEspecial\":\"Nenhuma\",\"localProvaUF\":\"MG\",\"localProvaMunicipio\":\"Belo Horizonte\"}"'
Using spec file: /home/luiza/mywork/spec.yaml
Minifab Execution Context:
    FABRIC_RELEASE=2.3.3
    CHANNEL_NAME=jornada-channel
    PEER_DATABASE_TYPE=golevel
    CHAINCODE_LANGUAGE=go
    CHAINCODE_NAME=enem
    CHAINCODE_VERSION=4.9
    CHAINCODE_INIT_REQUIRED=false
    CHAINCODE_PARAMETERS="StoreInscricao", "{\"NumInscricao\":\"numInc1\",\"CPF\":\"123.456.789-00\",\"dataNascimento\":\"1990-05-20T00:00:00
Z\",\"nome\":\"João da Silva\",\"nomePai\":\"Carlos da Silva\",\"nomeMae\":\"Maria da Silva\",\"sexo\":\"Masculino\",\"cor\":\"Pardo\",\"RG\"
:{\"numero\":\"MG-12.345.678\",\"orgaoExpedidor\":\"SSP\",\"UF\":\"MG\"},\"estadoCivil\":\"Solteiro\",\"nacionalidade\":\"Brasileiro\",\"natu
ralidade\":{\"codigoMunicipio\":\"3106200\",\"nomeMunicipio\":\"Belo Horizonte\",\"UF\":\"MG\",\"ehEstrangeiro\":false,\"nomeMunicipioEstrang
eiro\":\"\"},\"endereco\":\"Rua Exemplo, 123, Bairro Centro, Belo Horizonte - MG\",\"anoEdicao\":\"2024\",\"linguaEstrangeira\":\"Inglês\",\"
nivelEscolaridade\":\"Ensino Médio Completo\",\"tipoEnsinoMedio\":\"Regular\",\"instituicaoEnsinoMedio\":\"Escola Estadual Exemplo\",\"opcaoA
tendimentoEspecial\":\"Nenhuma\",\"localProvaUF\":\"MG\",\"localProvaMunicipio\":\"Belo Horizonte\"}"
    CHAINCODE_PRIVATE=false
    CHAINCODE_POLICY=
    TRANSIENT_DATA=
    APP_NAME=sample
    BLOCK_NUMBER=newest
    EXPOSE_ENDPOINTS=true
    CURRENT_ORG=org0.acadblock.br
    HOST_ADDRESSES=192.168.100.14
    CONFIG_MERGE_TYPE=app
    WORKING_DIRECTORY: /home/luiza/mywork
.....
# Preparing for the following operations: *********************
  verify options, cc invoke
.......
# Running operation: ******************************************
  verify options
..
# Running operation: ******************************************
  cc invoke
...............
# Chaincode invocation results ********************************
  ['\x1b[34m2024-06-17 00:22:27.807 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001\x1b[0m Chaincode invoke successful. result: status:
200 ']

# STATS *******************************************************
minifab: ok=34  failed=0

real    0m6.618s
user    0m5.950s
sys     0m0.892s
```

Fonte: autoria própria.

Após o registro da inscrição, visando não só testar a mesma, como também realizar a visualização das informações de uma inscrição, deve-se utilizar a função de leitura da inscrição, a readInscricaoByNumInscricao(). O comando segue o mesmo padrão que aquele utilizado na função de armazenar uma inscrição, porém, com parâmetros diferentes. Já que a função testada será outra, primeiro, deve-se passar o nome da nova funçao: ReadInscricaoByNumInscricao, seguido pelo número de inscrição da inscrição que será visualizada. Nesse caso, conforme já apresentado, tem-se o “numInc1”. Assim, o comando deve ser o seguinte:

```
./minifab invoke -p '"ReadInscricaoByNumInscricao",
"numInc1"'
```

Ao executar o comando, a função retornará, além do *“status: 200”*, e a mensagem vista na chamada anterior, o JSON, em formato de *string*, que representa a inscrição associada àquele número de inscrição. O retorno da chamada pode ser conferido na figura abaixo.

Figura 12 - Captura de tela da operação *readInscricaoByNumInscricao()*

Fonte: autoria própria.

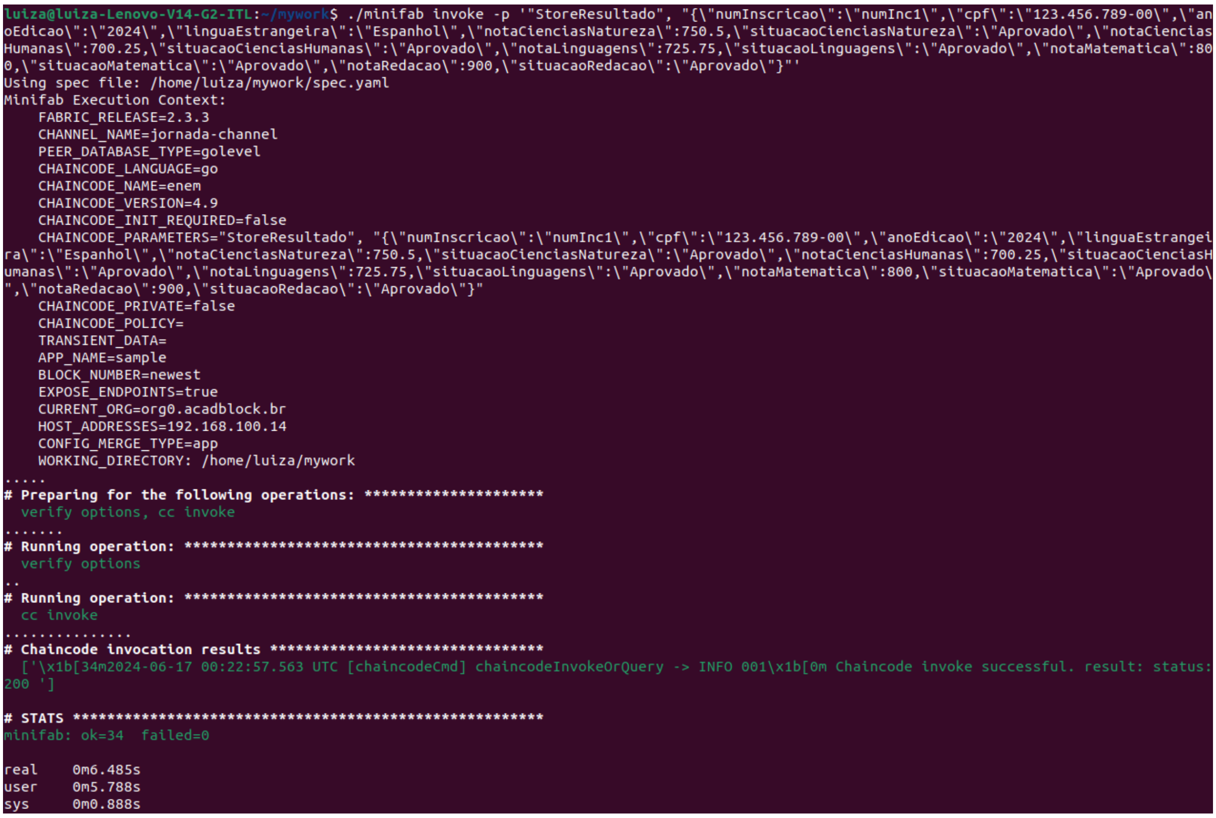
Assim, conclui-se a utilização das funções de inscrição, e pode-se iniciar a utilização das funções referentes ao resultado. O número de inscrição será o mesmo, representando a continuação da jornada do candidato ao longo do processo. Ou seja, simula-se o resultado do candidato que foi inscrito nos passos anteriores. Pode-se conferir abaixo o objeto que representa o resultado fictício do candidato, no formato JSON.

```
{
  "numInscricao": "numInc1",
  "cpf": "123.456.789-00",
  "anoEdicao": "2024",
  "linguaEstrangeira": "Espanhol",
  "notaCienciasNatureza": 750.5,
  "situacaoCienciasNatureza": "Aprovado",
  "notaCienciasHumanas": 700.25,
  "situacaoCienciasHumanas": "Aprovado",
  "notaLinguagens": 725.75,
  "situacaoLinguagens": "Aprovado",
  "notaMatematica": 800,
  "situacaoMatematica": "Aprovado",
  "notaRedacao": 900,
  "situacaoRedacao": "Aprovado"
}
```

Assim, ao formatar o objeto JSON no formato apropriado para que seja passado no terminal, o comando para armazenar o resultado na rede segue o padrão dos demais - primeiro, o nome da função, StoreResultado, seguido pelo objeto JSON no formato de string. O comando resultante, além do resultado da utilização do mesmo, pode ser conferido abaixo.

```
./minifab invoke -p '"StoreResultado", "{\"numInscricao\":\"numInc1\",\"cpf\":\"123.456.789-00\",\"anoEdicao\":\"2024\",\"linguaEstrangeira\":\"Espanhol\",\"notaCienciasNatureza\":750.5,\"situacaoCienciasNatureza\":\"Aprovado\",\"notaCienciasHumanas\":700.25,\"situacaoCienciasHumanas\":\"Aprovado\",\"notaLinguagens\":725.75,\"situacaoLinguagens\":\"Aprovado\",\"notaMatematica\":800,\"situacaoMatematica\":\"Aprovado\",\"notaRedacao\":900,\"situacaoRedacao\":\"Aprovado\"}"'
```

Figura 13 - Captura de tela da operação *storeResultado()*



Como pode-se observar, recebe-se uma mensagem de sucesso, indicando a ausência de erros na execução da função. Logo, o resultado foi devidamente armazenado na rede.

Para conferir o resultado, basta executar a função ReadResultadoByNumInscricao, seguido pelo número de inscrição o qual está atrelado ao resultado. Nesse caso, é o numInc1, seguindo a inscrição fictícia do candidato. Logo, o comando a ser executado encontra-se abaixo.

```
./minifab invoke -p '"ReadResultadoByNumInscricao", "numInc1"'
```

Após a execução do comando, no mesmo padrão do retorno do ReadInscricaoByNumInscricao, recebe-se uma mensagem de sucesso, com *status 200*, além de trazer, no *payload*, o objeto JSON que representa o resultado, em formato de *string*.

Figura 14 - Captura de tela da operação *ReadResultadoByNumInscricao()*

```
luiza@luiza-Lenovo-V14-G2-ITL:~/mywork$ ./minifab invoke -p '"ReadResultadoByNumInscricao", "numInc1"'
Using spec file: /home/luiza/mywork/spec.yaml
Minifab Execution Context:
    FABRIC_RELEASE=2.3.3
    CHANNEL_NAME=jornada-channel
    PEER_DATABASE_TYPE=golevel
    CHAINCODE_LANGUAGE=go
    CHAINCODE_NAME=enem
    CHAINCODE_VERSION=4.9
    CHAINCODE_INIT_REQUIRED=false
    CHAINCODE_PARAMETERS="ReadResultadoByNumInscricao", "numInc1"
    CHAINCODE_PRIVATE=false
    CHAINCODE_POLICY=
    TRANSIENT_DATA=
    APP_NAME=sample
    BLOCK_NUMBER=newest
    EXPOSE_ENDPOINTS=true
    CURRENT_ORG=org0.acadblock.br
    HOST_ADDRESSES=192.168.100.14
    CONFIG_MERGE_TYPE=app
    WORKING_DIRECTORY: /home/luiza/mywork
.....
# Preparing for the following operations: *********************
  verify options, cc invoke
.......
# Running operation: ******************************************
  verify options
..
# Running operation: ******************************************
  cc invoke
..............
# Chaincode invocation results ********************************
  ['\x1b[34m2024-06-17 00:23:37.787 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001\x1b[0m Chaincode invoke successful. result: status:
200 payload:"{\\"numInscricao\\":\\"numInc1\\",\\"cpf\\":\\"123.456.789-00\\",\\"anoEdicao\\":\\"2024\\",\\"linguaEstrangeira\\":\\"Espanhol\
\",\\"notaCienciasNatureza\\":750.5,\\"situacaoCienciasNatureza\\":\\"Aprovado\\",\\"notaCienciasHumanas\\":700.25,\\"situacaoCienciasHumanas
\\":\\"Aprovado\\",\\"notaLinguagens\\":725.75,\\"situacaoLinguagens\\":\\"Aprovado\\",\\"notaMatematica\\":800,\\"situacaoMatematica\\":\\"A
provado\\",\\"notaRedacao\\":900,\\"situacaoRedacao\\":\\"Aprovado\\"}" ']

# STATS *******************************************************
minifab: ok=34  failed=0

real    0m6.348s
user    0m5.628s
sys     0m0.901s
```

Assim, conclui-se com sucesso a jornada do candidato, que começa na inscrição, e termina ao se obter seu resultado.

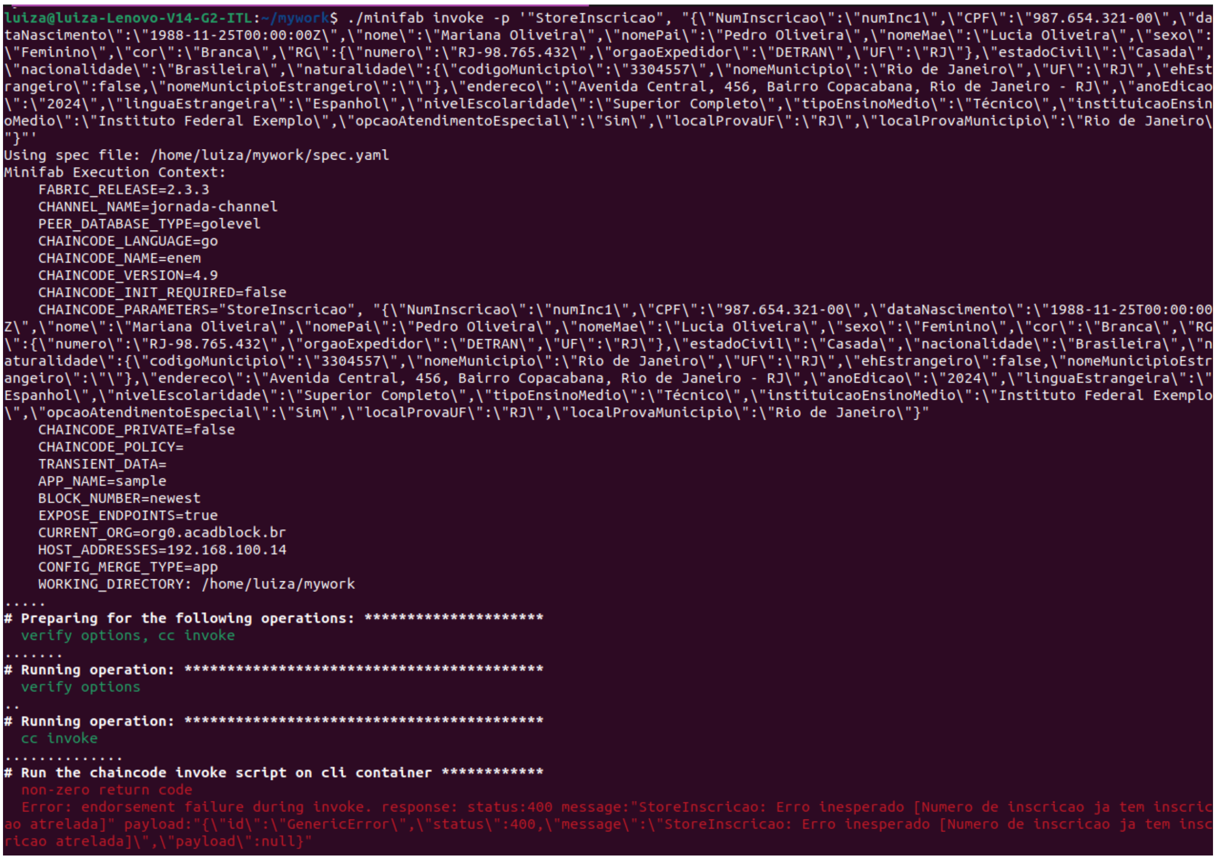
4.2 ERROS LEVANTADOS NAS VALIDAÇÕES

Embora o ideal seja que a jornada seja completamente bem sucedida, alguns erros são implementados para conferir ao código uma maior solidez. Nessa seção, serão forçados erros, ao utilizar exemplos de alguns casos indesejáveis no sistema, para demonstrar como esses casos serão gerenciados.

O primeiro caso que levanta um erro é quando já foi feita uma inscrição com um determinado número de inscrição, e é feita uma tentativa de realizar uma segunda inscrição para aquele mesmo número de inscrição. Foi criado um novo objeto JSON para representar uma inscrição, com informações totalmente novas, mas com o mesmo número de inscrição utilizado na seção 4.1. Assim, como pode-se observar na figura 14, ao executar o comando para armazenar essa inscrição, é gerado um erro, informando que já existe uma inscrição com o número de inscrição fornecido, e a inscrição não é armazenada na rede.

```
./minifab invoke -p '"StoreInscricao",
"{\"NumInscricao\":\"inc1\",\"CPF\":\"987.654.321-
00\",\"dataNascimento\":\"1991-06-15T00:00:00Z\",\
"nome\":\"Maria Oliveira\",\"nomePai\":\"José
Oliveira\",\"nomeMae\":\"Ana
Oliveira\",\"sexo\":\"Feminino\",\"cor\":\"Branco\
",\"RG\":{\"numero\":\"SP-99.888.777\",\"orgaoExpe
didor\":\"SSP\",\"UF\":\"SP\"},\"estadoCivil\":\"C
asada\",\"nacionalidade\":\"Brasileira\",\"natural
idade\":{\"codigoMunicipio\":\"3550308\",\"nomeMun
icipio\":\"São
Paulo\",\"UF\":\"SP\",\"ehEstrangeiro\":false,\"no
meMunicipioEstrangeiro\":\"\"},\"endereco\":\"Av.
Principal, 456, Bairro Novo, São Paulo -
SP\",\"anoEdicao\":\"2023\",\"linguaEstrangeira\":
\"Espanhol\",\"nivelEscolaridade\":\"Ensino Médio
Completo\",\"tipoEnsinoMedio\":\"Regular\",\"insti
tuicaoEnsinoMedio\":\"Escola Estadual
Nova\",\"opcaoAtendimentoEspecial\":\"Ledor\",\"lo
calProvaUF\":\"SP\",\"localProvaMunicipio\":\"São
Paulo\"}"'
```

Figura 15 - Captura de tela de erro no *storeInscricao()* de inscrição já existente



Outro caso de erro projetado é quando alguma função não recebe o número correto de parâmetros. Para isso, foi criado o seguinte comando:

```
./minifab invoke -p '"StoreInscricao", "ABCDE",
"12345"'
```

No exemplo abaixo, a função *StoreInscricao* é passada com dois parâmetros, sendo que a mesma só aceita um parâmetro. O retorno da função informa o erro, esclarecendo quantos parâmetros são aceitos pela função, e quantos parâmetros foram fornecidos.

Figura 16 - Captura de tela de erro no *storeInscricao()* com parâmetros insuficientes

Segundo as regras de negócio estabelecidas, um resultado não deverá ser registrado caso o número de inscrição contido nele não conte ainda com uma inscrição. Para demonstrar esse cenário, foi novamente criado um objeto JSON com um número de inscrição o qual não havia inscrição associada na rede. Logo, é levantado um erro, informando a impossibilidade de registrar um resultado sem inscrição associada. O erro ocasionado pelo comando abaixo pode ser visto na figura 16.

```
./minifab invoke -p '"StoreResultado",
"{\"numInscricao\":\"numeroSemInscricao\",\"cpf\":
\"123.456.789-00\",\"anoEdicao\":\"2024\",\"lingua
Estrangeira\":\"Espanhol\",\"notaCienciasNatureza\
":750.5,\"situacaoCienciasNatureza\":\"Aprovado\",
\"notaCienciasHumanas\":700.25,\"situacaoCienciasH
umanas\":\"Aprovado\",\"notaLinguagens\":725.75,\"
situacaoLinguagens\":\"Aprovado\",\"notaMatematica
```

```
\":800,\"situacaoMatematica\":\"Aprovado\",\"notaR
edacao\":900,\"situacaoRedacao\":\"Aprovado\"}"'
```

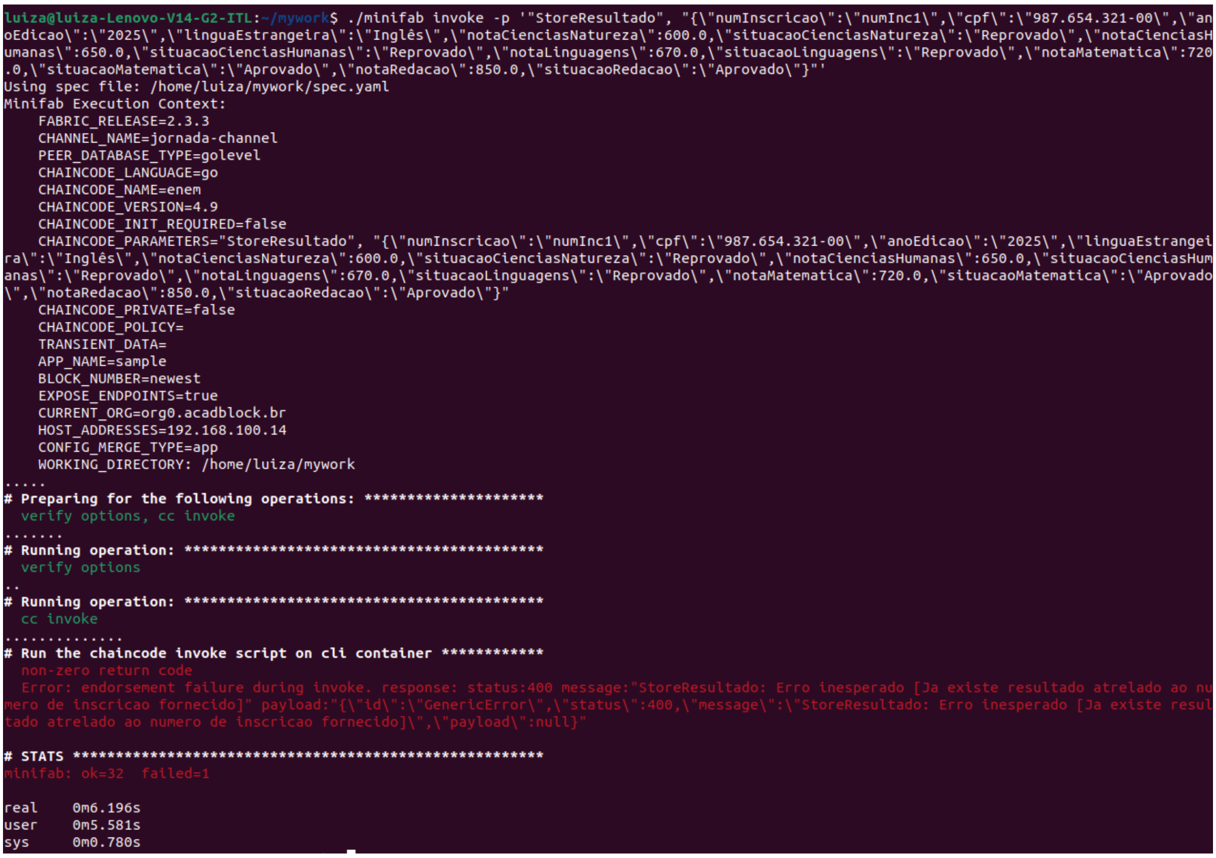
Figura 17 - Captura de tela de erro no *storeResultado()* quando não há inscrição correspondente

```
luiza@luiza-Lenovo-V14-G2-ITL:~/mywork$ ./minifab invoke -p '"StoreResultado", "{\"numInscricao\":\"numeroSemInscricao\",\"cpf\":\"123.456.78
9-00\",\"anoEdicao\":\"2024\",\"linguaEstrangeira\":\"Espanhol\",\"notaCienciasNatureza\":750.5,\"situacaoCienciasNatureza\":\"Aprovado\",\"n
otaCienciasHumanas\":700.25,\"situacaoCienciasHumanas\":\"Aprovado\",\"notaLinguagens\":725.75,\"situacaoLinguagens\":\"Aprovado\",\"notaMate
matica\":800,\"situacaoMatematica\":\"Aprovado\",\"notaRedacao\":900,\"situacaoRedacao\":\"Aprovado\"}"'
Using spec file: /home/luiza/mywork/spec.yaml
Minifab Execution Context:
    FABRIC_RELEASE=2.3.3
    CHANNEL_NAME=jornada-channel
    PEER_DATABASE_TYPE=golevel
    CHAINCODE_LANGUAGE=go
    CHAINCODE_NAME=enem
    CHAINCODE_VERSION=4.9
    CHAINCODE_INIT_REQUIRED=false
    CHAINCODE_PARAMETERS="StoreResultado", "{\"numInscricao\":\"numeroSemInscricao\",\"cpf\":\"123.456.789-00\",\"anoEdicao\":\"2024\",\"ling
uaEstrangeira\":\"Espanhol\",\"notaCienciasNatureza\":750.5,\"situacaoCienciasNatureza\":\"Aprovado\",\"notaCienciasHumanas\":700.25,\"situac
aoCienciasHumanas\":\"Aprovado\",\"notaLinguagens\":725.75,\"situacaoLinguagens\":\"Aprovado\",\"notaMatematica\":800,\"situacaoMatematica\":
\"Aprovado\",\"notaRedacao\":900,\"situacaoRedacao\":\"Aprovado\"}"
    CHAINCODE_PRIVATE=false
    CHAINCODE_POLICY=
    TRANSIENT_DATA=
    APP_NAME=sample
    BLOCK_NUMBER=newest
    EXPOSE_ENDPOINTS=true
    CURRENT_ORG=org0.acadblock.br
    HOST_ADDRESSES=192.168.100.14
    CONFIG_MERGE_TYPE=app
    WORKING_DIRECTORY: /home/luiza/mywork
.....
# Preparing for the following operations: *********************
  verify options, cc invoke
.......
# Running operation: ******************************************
  verify options
..
# Running operation: ******************************************
  cc invoke
.............
# Run the chaincode invoke script on cli container ************
  non-zero return code
  Error: endorsement failure during invoke. response: status:400 message:"StoreResultado: Erro inesperado [Nao ha registro de inscricao atrel
ado ao numero de inscricao fornecido, logo, nao pode ser cadastrado um resultado]" payload:"{\"id\":\"GenericError\",\"status\":400,\"message
\":\"StoreResultado: Erro inesperado [Nao ha registro de inscricao atrelado ao numero de inscricao fornecido, logo, nao pode ser cadastrado u
m resultado]\",\"payload\":null}"

# STATS *******************************************************
minifab: ok=32  failed=1

real    0m6.232s
user    0m5.687s
sys     0m0.713s
```

Ainda segundo as regras de negócio, deve existir apenas um resultado para um mesmo número de inscrição, não podendo realizar o registro de um resultado se aquele número de inscrição já tiver resultado atrelado. Assim, quando se tenta registrar um novo resultado para o numInscricao “numInc1”, através do comando abaixo, o erro da figura 18 é lançado, informando o equívoco.

```
./minifab    invoke    -p    '"StoreResultado",
"{\"numInscricao\":\"numInc1\",\"cpf\":\"987.654.3
21-00\",\"anoEdicao\":\"2025\",\"linguaEstrangeira
\":\"Inglês\",\"notaCienciasNatureza\":600.0,\"sit
uacaoCienciasNatureza\":\"Reprovado\",\"notaCienci
asHumanas\":650.0,\"situacaoCienciasHumanas\":\"Re
provado\",\"notaLinguagens\":670.0,\"situacaoLingu
agens\":\"Reprovado\",\"notaMatematica\":720.0,\"s
ituacaoMatematica\":\"Aprovado\",\"notaRedacao\":8
50.0,\"situacaoRedacao\":\"Aprovado\"}"'
```

Figura 18: Captura de tela de erro no *storeInscricao()*



Assim, conclui-se a demonstração das principais validações envolvidas na etapa do desenvolvimento.

5. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Conforme apresentado no capítulo 1, o objetivo da pesquisa foi desenvolver um MVP de uma *chaincode* para fazer a rastreabilidade da jornada do candidato do ENEM, dentro do contexto do projeto da Jornada do Estudante. A *chaincode* foi projetada e implementada, usando as tecnologias já presentes na rede do Jornada do Estudante existente hoje.

Foi feita uma pesquisa para entender os principais processos, estruturas, e informações envolvidas ao longo da jornada, para que a solução refletisse a realidade do modelo atual de forma mais próxima possível, dentro do contexto de um MVP.

As estruturas envolvidas no processo, e as regras de negócio foram definidas, para embasar o desenvolvimento do código. Uma nova *chaincode* foi desenvolvida para incrementar o código fonte do Jornada do Estudante, agindo como um MVP de uma rede blockchain que faça o registro dos dados de um candidato e da sua inscrição no processo do ENEM. Assim, é possível registrar e consultar os dados referentes à inscrição e ao resultado do candidato na prova.

Após a conclusão da etapa de desenvolvimento, os testes foram feitos para garantir o respeito às regras de negócio e o funcionamento correto da solução. Todas as funções funcionaram conforme o esperado, implementando as validações necessárias para garantir o cumprimento do fluxo natural do candidato ao longo da sua trajetória no processo do ENEM. As contribuições desse trabalho, então, podem ser enumeradas:

(1) criação de uma arquitetura blockchain para acompanhamento do candidato no processo do ENEM;
(2) a centralização de informações e dados quanto ao processo de inscrição, obtenção de resultado, e as informações envolvidas na jornada do candidato;
(3) implementação de uma *chaincode* para armazenamento e leitura das principais etapas envolvidas ao longo do processo de inscrição e obtenção do resultado da prova;
(4) entrega de um MVP para fazer o acompanhamento de inscrições no ENEM.

Dos possíveis trabalhos a serem desenvolvidos a partir deste, podem-se sugerir os seguintes:

(1) a integração da estrutura atual com a estrutura existente de registro de Alunos, no Jornada do Estudante. Assim, seria possível a recuperação de dados fornecidos em inscrições passadas do ENEM, diminuindo a quantidade de informações demandadas dos estudantes ao longo do processo de inscrição, tornando-o menos massante, e aprimorando a experiência do usuário. Além disso, isso resultaria em uma maior

rastreabilidade na vida do candidato enquanto aluno, gerando maior possibilidade de análise de coerência das informações ao longo dos anos, e, inclusive, podendo facilitar alguns processos;

(2) a inclusão do questionário socioeconômico na estrutura atual, tornando-o recuperável entre edições, facilitando a experiência do usuário no momento da inscrição, e centralizando os dados do questionário socioeconômico, visando garantir maior facilidade para a criação de estudos e/ou políticas públicas;

(3) a criação de uma interface gráfica, para facilitar o uso e a demonstração da solução, além de alterações no código para facilitar o envio de informações em volume para a aplicação.

REFERÊNCIAS

ALHARBY, Maher ; MOORSEL, Aad van. **Blockchain Based Smart Contracts : A Systematic Mapping Study.** Computer Science & Information Technology (CS & IT), 2017. Disponível em: <https://arxiv.org/abs/1710.06372>. Acesso em: 8 dez. 2023.

ARAUJO, Beatriz. Quem criou o Fies, SiSU e ProUni? Saiba mais. **Terra**. Disponível em: <https://www.terra.com.br/noticias/educacao/quem-criou-o-fies-sisu-e-prouni-saiba-mais,d1120f92d9e97eba2fc89330e4dc8d27fikee7yn.html>. Acesso em: 8 dez. 2023.

BATISTA, R. Enem 20 anos: a transformação da maior prova do Brasil. **UOL.** Disponível em: <https://vestibular.brasilescola.uol.com.br/enem/enem-20-anos-transformacao-maior-prova-brasil.htm>. Acesso em: 13 nov. 2023.

CAMPOS, LORRAINE. Quantas questões tem o Enem e como funciona a prova. **Vestibular Brasil Escola.** Disponível em: <https://vestibular.brasilescola.uol.com.br/enem/quantas-questoes-tem-enem-como-funciona-prova.htm>. Acesso em: 4 dez. 2023.

ENEM. **Gov.br**. Disponível em: <https://www.gov.br/inep/pt-br/areas-de-atuacao/avaliacao-e-exames-educacionais/enem>. Acesso em: 13 nov. 2023.

ENEM e Sisu revolucionaram a educação no País e devem agora ser aprimorados, diz pesquisador. **Estadão**. Disponível em: <https://www.estadao.com.br/educacao/enem-e-sisu-revolucionaram-a-educacao-no-pais-e-devem-agora-ser-aprimorados-diz-pesquisador/>. Acesso em: 4 dez. 2023.

ESPECIALISTAS repercutem recorde histórico de baixa adesão ao Enem em 2021. **Fundação Telefônica Vivo.** Disponível em: <https://www.fundacaotelefonicavivo.org.br/noticias/especialistas-repercutem-recorde-historico-de-baixa-adesao-ao-enem-em-2021/>. Acesso em: 4 dez. 2023.

FUNDO de financiamento ao estudante do ensino superior. **Mec.gov.br.** Disponível em: <https://sisfiesportal.mec.gov.br/?pagina=condicoes#:~:text=Durante%20o%20curso%3A%20Pagamento%20trimestral,per%C3%ADodo%20financiado%20do%20curso)%5D.>. Acesso em: 8 dez. 2023.

GIDEÃO, Leonardo. **Blockchain Aplicada em uma Carteira Digital Acadêmica para Facilitar o Controle Curricular Estudantil.** 2023. UFSC, Florianópolis, 2023.

HISTÓRICO. **Gov.br**. Disponível em: <https://www.gov.br/inep/pt-br/areas-de-atuacao/avaliacao-e-exames-educacionais/enem/historico>. Acesso em: 4 dez. 2023.

HYPERLEDGER. Minifabric: A Hyperledger Fabric Quick Start Tool (with Video Guides) – Hyperledger Foundation. **Hyperledger.org.** Disponível em: <https://www.hyperledger.org/blog/2020/04/29/minifabric-a-hyperledger-fabric-quick-start-tool-with-video-guides>. Acesso em: 13 maio 2024.

JOE ABOU JAOUDE; RAAFAT GEORGE SAADÉ. **Blockchain Applications – Usage in Different Domains.** IEEE Access, v. 7, p. 45360–45381, 2019. **IEEE**. Disponível em: <https://ieeexplore.ieee.org/document/8656511>. Acesso em: 14 nov. 2023.

JOSA, Lucas. O que são smart contracts? **Exame**. Disponível em: <https://exame.com/future-of-money/o-que-sao-smart-contracts/>. Acesso em: 8 dez. 2023.

Liu, B.; Si, X.; Kang, H. **A Literature Review of Blockchain-Based Applications in Supply Chain.** *Sustainability* **2022**, *14*, 15210. Disponível em: <https://doi.org/10.3390/su142215210>. Acesso em: 14 nov. 2023.

MERKLE, Ralph C. **One Way Hash Functions and DES.** Lecture Notes in Computer Science, p. 428–446, 1990. Disponível em: <https://link.springer.com/chapter/10.1007/0-387-34805-0_40>. Acesso em: 14 nov. 2023.

MILOJICIC, Dejan; KALOGERAKI, Vana; LUKOSE, Rajan; *et al*. **Peer-to-Peer Computing.** [s.l.: s.n.], 2003. Disponível em: <https://www.cs.kau.se/cs/education/courses/dvad02/p2/seminar4/Papers/HPL-2002-57R1.pdf>. Acesso em: 9 dez. 2023.

O que é blockchain e como funciona a tecnologia por trás do bitcoin? **Exame.** Disponível em: <https://exame.com/invest/guia/o-que-e-blockchain-e-como-funciona-a-tecnologia-por-tras-do-bitcoin/>. Acesso em: 14 nov. 2023.

O que é blockchain na AWS? **Amazon Web Services, Inc.** Disponível em: <https://aws.amazon.com/pt/what-is/blockchain/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc>. Acesso em: 8 dez. 2023.

O que é Peer-to-Peer (P2P)? Entenda a relação dessa tecnologia com criptos. **InfoMoney.** Disponível em: <https://www.infomoney.com.br/guias/peer-to-peer-p2p/>. Acesso em: 9 dez. 2023.

PORTAL Único de Acesso ao Ensino Superior. **Mec.gov.br.** Disponível em: <https://acessounico.mec.gov.br/acesso-unico>. Acesso em: 5 dez. 2023.

PROUNI. **Mec.gov.br.** Disponível em: <https://acessounico.mec.gov.br/prouni>. Acesso em: 5 dez. 2023.

PROUNI - Apresentação. **Mec.gov.br**. Disponível em: <http://portal.mec.gov.br/index.php?option=com_content&view=article&id=205&Itemid=298&msg=1&l=aW5kZXgucGhwP29wdGlvbj1jb21fY29udGVudCZ2aWV3PWJ1c2NhZ2VyYWwmSXRlbWlkPTE2NCZwYXJhbXNbc2VhcmNoX3JlbGV2YW5jZV09cHJvdW5pJmQ9cyZwYXJhbXNbZGVdPSZwYXJhbXNbYXRlXT0mcGFyYW1zW2NhdGlkXT0mcGFyYW1zW3NlYXJjaF9tZXRob2RdPWFsbCZwYXJhbXNbb3JkXT1wcg==>. Acesso em: 5 dez. 2023.

PUBLIC Keys, Private Keys, and Certificates (Configuring Java CAPS for SSL Support). **Oracle.com.** Disponível em: <https://docs.oracle.com/cd/E19509-01/820-3503/ggbgc/index.html>. Acesso em: 9 dez. 2023.

REDES Par-a-Par (P2P). **Ufrj.br.** Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/p2p/>. Acesso em: 9 dez. 2023.

SMART contracts: entenda o que são e qual a relação com as criptomoedas. **CNN Brasil.** Disponível em: <https://www.cnnbrasil.com.br/economia/smart-contracts/>. Acesso em: 8 dez. 2023.

SOBTI, R.; GEETHA, G. **Cryptographic Hash Functions: A Review**. IJCSI International Journal of Computer Science Issues. Disponível em: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=85abc4805adb741b0f8c962794d2ab4dac975c5f>. Acesso em: 14 nov. 2023.

TUDO sobre o Enem 2023: o que pode levar e o que costuma cair na prova**. Globo**. Disponível em: <https://g1.globo.com/educacao/enem/2023/noticia/2023/10/05/tudo-sobre-o-enem-2023-o-que-levar-o-que-costuma-cair-na-prova-e-dicas-para-redacao-nota-mil.ghtml>. Acesso em: 13 nov. 2023.

TYPES of Blockchains Explained- Public Vs. Private Vs. Consortium. **Blockchain-council.org.** Disponível em: <https://www.blockchain-council.org/blockchain/types-of-blockchains-explained-public-vs-private-vs-consortium/#:~:text=Public%20Blockchain%3A%20Open%20and%20permissionless,Hyperledger%20Fabric%20and%20R3%20Corda.>. Acesso em: 9 dez. 2023.

TYPES of Blockchain: Public, Private, or Something in Between | **JD Supra.** JD Supra. Disponível em: <https://www.jdsupra.com/legalnews/types-of-blockchain-public-private-or-5282575/>. Acesso em: 9 dez. 2023.

UMA plataforma de blockchain para a empresas. **hyperledger-fabric.readthedocs.io**. Diponível em <https://hyperledger-fabric.readthedocs.io/pt/latest/>. Acesso em: 13 maio 2024.

WHAT are smart contracts on blockchain? | IBM. **Ibm.com.** Disponível em: <https://www.ibm.com/topics/smart-contracts>. Acesso em: 8 dez. 2023.

WHITAKER, A. **Art and Blockchain: A Primer, History, and Taxonomy of Blockchain Use Cases in the Arts.** Disponível em: <https://www.researchgate.net/publication/337064517_Art_and_Blockchain_A_Primer_History_and_Taxonomy_of_Blockchain_Use_Cases_in_the_Arts>. Acesso em: 13 nov. 2023.

YAGA, Dylan; MELL, Peter; ROBY, Nik; *et al*. **Blockchain Technology Overview.** National Institute of Standards and Technology, v. 1, n. 1, 2018.
RODECK, David ; CURRY, Benjamin. What Is Blockchain? [s.l.: s.n.], 2022. Disponível em: <https://communications.pasenategop.com/wp-content/uploads/sites/15/2022/06/What-Is-Blockchain.pdf>. Acesso em: 9 dez. 2023.

ZHANG, Shijie ; JONG HYOUK LEE. **Analysis of the main consensus protocols of blockchain.** ICT Express, v. 6, n. 2, p. 93–97, 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S240595951930164X>. Acesso em: 9 dez. 2023.

3,9 milhões estão inscritos no Enem 2023. **Gov.br**. Disponível em: <https://www.gov.br/inep/pt-br/assuntos/noticias/enem/3-9-milhoes-estao-inscritos-no-enem-2023>. Acesso em: 2 dez. 2023.

Rastreabilidade do ENEM na Jornada do Estudante

Luiza Domingues Medeiros, Jean Everson Martina

Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil

luizadmeds@gmail.com, jeanmartina@gmail.com

Abstract: *Blockchain technology enables the immutable, and decentralized storage of information, recording each transaction highly reliably. Currently, it is understood that there are intrinsic benefits to blockchain that make it advantageous to be applied in various areas. In Brazil, many students seeking admission to Higher Education Institutions must take the National High School Exam (ENEM), as it is a selection criterion for the main public admission programs in HEIs. Thus, the success and reliability of these processes depend on the integrity of the results obtained from the ENEM. In this context, this work proposes the application of blockchain to record and track the movements of candidates of the ENEM, aiming to ensure greater traceability and security in the process through the characteristics of blockchains*.

Resumo: *A tecnologia de blockchain permite o armazenamento de informações de forma imutável, distribuída, e descentralizada, registrando cada transação que acontece de forma altamente confiável. No Brasil, muitos alunos que procuram ingresso em uma Instituição de Ensino Superior (IES) deverão realizar o Exame Nacional do Ensino Médio (ENEM), uma vez que este é critério de seleção dos principais programas de ingresso em IES. Assim, o sucesso desses processos dependem da integridade dos resultados obtidos a partir do ENEM. Nesse contexto, propõe-se a aplicação de uma blockchain para fazer o registro e acompanhamento de movimentações de candidatos do ENEM, visando garantir, por meio das características das blockchains, maior rastreabilidade e segurança no processo.*

Introdução

A confiança nas informações armazenadas em computadores sempre foi uma preocupação. Stuart Haber e Scott Stornetta, pioneiros da criptografia, questionaram a integridade desses dados e a confiabilidade de uma autoridade central para gerenciá-los (WHITAKER, 2019). Assim, desenvolveram um sistema criptografado com registros interligados por timestamps, formando a base do que conhecemos hoje como blockchain. Este conceito se consolidou em 1991 com o artigo "How to Time-Stamp a Digital Document" (HABER; STORNETTA, 1991). Em 2003, fundaram a Surety, oferecendo armazenamento seguro para cientistas e publicando semanalmente no The New York Times um código de verificação para garantir a integridade dos dados (WHITAKER, 2019).

A blockchain ganhou notoriedade em 2009 com a criação do Bitcoin por Satoshi Nakamoto, que adicionou incentivos financeiros ao resolver problemas matemáticos (WHITAKER, 2019). Em 2014, Vitalik Buterin introduziu o Ethereum, permitindo a execução de contratos inteligentes e a tokenização (WHITAKER, 2019). A tecnologia blockchain, baseada na descentralização e imutabilidade, tem potencial transformador em diversos contextos, sendo considerada uma das tecnologias mais revolucionárias da atualidade (EXAME, 2022).

Pesquisas recentes destacam o crescente interesse na aplicação de blockchain em diferentes setores, como demonstrado por Jaoude e Saadé (2019) e Liu, Si, e Kang (2022). No contexto brasileiro, o Exame Nacional de Ensino Médio (ENEM) é crucial para o ingresso no ensino superior (Ministério da Educação, 2023). Com milhões de estudantes participando anualmente, a transparência e segurança no processo são essenciais.

O Projeto Jornada do Estudante já utiliza blockchain para gerenciar diplomas e históricos escolares. Propõe-se, então, a extensão dessa tecnologia ao ENEM, aprimorando a credibilidade e rastreabilidade das notas e processos seletivos, beneficiando tanto os estudantes quanto o acompanhamento institucional do MEC.

Desenvolvimento do protótipo

Devido à falta de informações prévias disponibilizadas pelo INEP, foram utilizadas estratégias como a simulação de pedido de isenção de taxa para o ENEM 2024 e a pesquisa de dados solicitados em edições anteriores. Com isso, foi possível identificar os dados relevantes para incluir no protótipo, em um primeiro momento, e, então, definir melhor o objetivo dos contratos.

O objetivo é projetar contratos inteligentes referentes ao candidato inscrito e ao resultado do ENEM. O contrato referente ao candidato inscrito registrará a inscrição e os dados relevantes na blockchain, permitindo a localização da inscrição através do número de inscrição do candidato. O contrato referente ao resultado do candidato desempenhará a mesma função, registrando o resultado na blockchain e permitindo sua localização pelo número de inscrição.

Uma das metas deste trabalho é integrar a chaincode ao ambiente atual do projeto Jornada, exigindo a instalação do ambiente do Jornada para desenvolver na própria rede. Após configurar as variáveis de ambiente do Minifabric e instalar o código-fonte do Jornada na pasta "mywork", conforme padrões de laboratório e documentação do Minifabric, o ambiente foi preparado para instalação. Utilizando um script automatizado do projeto, todo o processo foi executado para inicializar a rede base, as chaincodes, aplicações, APIs e containers Docker, deixando a rede pronta para uso.

Para iniciar o desenvolvimento das chaincodes, foram definidos dois assets principais com seus atributos necessários: Inscricao e Result, formatados em JSON devido à sua ampla utilização e simplicidade de entrada de dados. Além disso, foi criada a estrutura Enem.

Após estruturar os assets, foram desenvolvidas funções baseadas nos objetivos definidos para o protótipo. Todas as funções estão encapsuladas na estrutura "Enem", conforme detalhado anteriormente. São quatro funções principais implementadas na chaincode, cada uma com regras de negócio específicas para garantir a consistência dos dados.

A função storeInscricao() é responsável por registrar uma nova inscrição na blockchain. Ela recebe como parâmetros um objeto JSON externo contendo os dados da inscrição (args) e uma interface (stub) para interação com o ledger. Inicialmente, o JSON é deserializado para a estrutura Inscricao. Se ocorrer algum erro durante essa transformação, a execução é interrompida e um erro é retornado. Após a conversão, a função gera uma chave única usando o prefixo "INSCRICAO_" seguido pelo número de inscrição. Antes de armazenar, é verificado se já existe uma inscrição com a mesma chave na blockchain. Em caso positivo, um erro é lançado e a função é encerrada. Caso contrário, a inscrição é convertida novamente para JSON e armazenada no ledger com a chave única. Se todas as etapas forem concluídas com sucesso, a função retorna nulo, indicando êxito na operação. Além disso, uma mensagem de sucesso é registrada nos logs dos peers.

A função readInscricaoByNumInscricao() tem como objetivo buscar uma inscrição na blockchain com base no número de inscrição fornecido. Ela recebe como parâmetros a interface (stub) para interação com o ledger e o número de inscrição. A função constrói a chave de busca concatenando o prefixo "INSCRICAO_" com o número de inscrição. Em seguida, utiliza o método stub.GetState() para recuperar os dados associados àquela chave do ledger. O método retorna um JSON e um erro. Se o erro não for nulo, ele é retornado indicando que ocorreu um problema durante a busca. Se o JSON retornado for nulo, significa que não há inscrição registrada com o número fornecido, e uma mensagem de aviso é retornada informando sobre a ausência da inscrição. Caso contrário, o JSON é convertido para o formato apropriado e retornado, permitindo a visualização dos dados da inscrição.

A função storeResultado() tem como objetivo armazenar um resultado na blockchain. Recebe como parâmetros a interface stub para interação com o ledger e args, contendo os dados do resultado a serem armazenados. Primeiramente, os dados são convertidos para a estrutura de resultado. Em seguida, é montada a chave de busca e de armazenamento concatenando o prefixo "RESULTADO_" com o número de inscrição presente nos dados do resultado. Se todas as validações são bem-sucedidas, os dados do resultado são armazenados utilizando a chave definida no ledger. Caso não haja erros durante o processo, a função não retorna nenhum valor, indicando que o armazenamento foi realizado com sucesso.

Por fim, a função readResultadoByNumInscricao() visa recuperar um resultado da blockchain com base no número de inscrição fornecido. Recebe como parâmetros a interface stub para interação com o ledger e o número de inscrição (numInscricao) como identificador único. Inicia-se montando a chave de busca concatenando "RESULTADO_" com o número de inscrição. Em seguida, utiliza-se stub.GetState() para buscar os dados associados à chave no ledger. Retorna um JSON com as informações do resultado se existir para o número de inscrição. Em caso de erro durante a busca ou se não houver resultado associado, retorna o erro correspondente ou uma mensagem de inexistência do resultado.

Considerações sobre o desenvolvimento

Durante o desenvolvimento das funções para interação com a blockchain, enfrentaram-se desafios significativos. A implementação em uma linguagem nova e a complexidade do MiniFabric foram obstáculos principais. A falta de clareza nas mensagens de erro do MiniFabric complicou a depuração, exigindo esforços adicionais para resolver problemas. Além disso, o processo de atualização do código no MiniFabric foi trabalhoso, requerendo especificar a chaincode e uma nova versão, o que prolongou os testes de alterações.

Um problema peculiar surgiu com as chaves de busca na blockchain. Inicialmente, ao usar o número de inscrição como chave, encontraram-se inconsistências nos resultados após consultas subsequentes. Isso ocorreu devido à reutilização da mesma chave, levando a dados corrompidos. Após pesquisa e ajustes, prefixar a chave com o tipo de objeto resolveu o problema, garantindo buscas precisas e consistentes.

Essas dificuldades foram superadas com esforço e pesquisa extensa, resultando em melhorias significativas no processo de desenvolvimento e implementação na blockchain.

Testes

Com o intuito de fazer uma demonstração clara, simulou-se a jornada do candidato ao longo do processo do ENEM, começando pela inscrição e depois pelo resultado. Para essa simulação, foram criados dois objetos JSON fictícios, representando uma inscrição e um resultado. Ambos os objetos utilizam o mesmo número de inscrição, refletindo o acompanhamento contínuo do candidato.

Após registrar a inscrição utilizando a função storeInscricao(), obteve-se confirmação de que a inscrição foi armazenada com sucesso na rede. Em seguida, utilizando a função readInscricaoByNumInscricao(), foi possível visualizar as informações da inscrição associada ao número específico.

Para concluir a demonstração da jornada do candidato no processo do ENEM, após registrar a inscrição utilizando a função `StoreInscricao`, segue-se com o registro do resultado do mesmo candidato. Utiliza-se o mesmo número de inscrição para manter a continuidade do processo simulado.

O comando para armazenar o resultado na rede, no formato padrão utilizado, é `StoreResultado` seguido pelo objeto JSON no formato de string. Após a execução bem-sucedida, uma mensagem de confirmação é recebida, indicando que o resultado foi devidamente armazenado na blockchain.

Para visualizar o resultado, utiliza-se a função `ReadResultadoByNumInscricao`, passando o número de inscrição associado ao resultado. Este comando permite acessar e verificar as informações do resultado registrado.

Esse processo demonstra como as funções de chaincode podem ser utilizadas para registrar e consultar informações de inscrição e resultado de candidatos na rede blockchain do ENEM, seguindo as regras de negócio estabelecidas.

Este processo ilustra como as funções de registro e leitura são utilizadas na chaincode para gerenciar dados de inscrições, demonstrando a interação com o MiniFabric no contexto do projeto.

No mais, casos de testes também foram feitos para garantir o comportamento de acordo com o esperado pelas regras estabelecidas.

Conclusão

Para concluir, o desenvolvimento deste trabalho focou na criação de um MVP de uma chaincode para rastrear a jornada do candidato do ENEM dentro do contexto do projeto Jornada do Estudante. Utilizando tecnologias existentes na rede atual, o objetivo foi integrar-se de forma precisa aos processos já estabelecidos. Isso envolveu uma pesquisa detalhada para entender os fluxos de inscrição, obtenção de resultados e as estruturas de dados envolvidas.

A implementação da chaincode foi guiada pela necessidade de registrar e consultar informações críticas, como dados de inscrição e resultados de prova, garantindo a conformidade com as regras de negócio estabelecidas. Os testes realizados asseguraram que todas as funcionalidades operassem corretamente, validando o cumprimento dos fluxos esperados e a integridade dos dados manipulados.

As contribuições deste trabalho incluem a criação de uma arquitetura blockchain específica para o acompanhamento do candidato no ENEM, centralizando e tornando acessíveis informações cruciais para os processos de inscrição e resultados. Além disso, a entrega de um MVP funcional proporciona uma base sólida para futuras iterações e melhorias no sistema.

Para trabalhos futuros, recomenda-se a integração mais profunda com o sistema de registro de alunos já existente, visando maior eficiência na recuperação de dados históricos de inscrições. Também é sugerida a inclusão do questionário socioeconômico na estrutura para facilitar o processo de inscrição e permitir análises mais abrangentes. Adicionalmente, o desenvolvimento de uma interface gráfica e ajustes no código para otimizar o desempenho e a escalabilidade são cruciais para tornar a solução mais acessível e robusta para seus usuários finais.

Referências

Liu, B.; Si, X.; Kang, H. A Literature Review of Blockchain-Based Applications in Supply Chain. *Sustainability* 2022, *14*, 15210. Disponível em: <https://doi.org/10.3390/su14 2215210>. Acesso em: 14 nov. 2023.

JOE ABOU JAOUDE; RAAFAT GEORGE SAADÉ. Blockchain Applications – Usage in Different Domains. IEEE Access, v. 7, p. 45360–45381, 2019. IEEE. Disponível em: <https://ieeexplore.ieee.org/document/8656511>. Acesso em: 14 nov. 2023.

O que é blockchain e como funciona a tecnologia por trás do bitcoin? **Exame.** Disponível em: <https://exame.com/invest/guia/o-que-e-blockchain-e-como-funciona-a-tecnologia-por-tras-do-bitcoin/>. Acesso em: 14 nov. 2023.

WHITAKER, A. Art and Blockchain: A Primer, History, and Taxonomy of Blockchain Use Cases in the Arts. Disponível em: <https://www.researchgate.net/publication/337064517_Art_and_Blockchain_A_Primer_History_and_Taxonomy_of_Blockchain_Use_Cases_in_the_Arts>. Acesso em: 13 nov. 2023.

```
package main

import (
   "encoding/json"
   "fmt"
   "time"

   customErrors "chaincodeErrors"


"github.com/hyperledger/fabric-chaincode-go/shim"
   "github.com/hyperledger/fabric-protos-go/peer"
)

type Inscricao struct {
      NumInscricao                          string
`json:"numInscricao"`
   CPF                       string    `json:"CPF"`
      DataNascimento                      time.Time
`json:"dataNascimento"`
     Nome                                   string
`json:"nome"`
     NomePai                                string
`json:"nomePai"`
     NomeMae                                string
`json:"nomeMae"`
     Sexo                                   string
`json:"sexo"`
   Cor                       string    `json:"cor"`
   RG                        RG        `json:"RG"`
      EstadoCivil                           string
`json:"estadoCivil"`
      Nacionalidade                         string
`json:"nacionalidade"`
      Naturalidade                       Municipio
`json:"naturalidade"`
```

```
	Endereco                              string
`json:"endereco"`
	AnoEdicao                             string
`json:"anoEdicao"`
	LinguaEstrangeira                     string
`json:"linguaEstrangeira"`
	Escolaridade                          string
`json:"nivelEscolaridade"`
	TipoEnsinoMedio                       string
`json:"tipoEnsinoMedio"`
	InstituicaoEnsinoMedio                string
`json:"instituicaoEnsinoMedio"`
	OpcaoAtendimentoEspecial              string
`json:"opcaoAtendimentoEspecial"`
	LocalProvaUF                          string
`json:"localProvaUF"`
	LocalProvaMunicipio                   string
`json:"localProvaMunicipio"`
}

type Municipio struct {
	CodigoMunicipio                       string
`json:"codigoMunicipio"`
	NomeMunicipio                         string
`json:"nomeMunicipio"`
	UF                        string `json:"UF"`
	EhEstrangeiro                         bool
`json:"ehEstrangeiro"`
	NomeMunicipioEstrangeiro              string
`json:"nomeMunicipioEstrangeiro"`
}

type Result struct {
	NumInscricao                          string
`json:"numInscricao"`
	CPF                       string  `json:"cpf"`
```

```
	AnoEdicao                        string
`json:"anoEdicao"`
	LinguaEstrangeira                string
`json:"linguaEstrangeira"`
	NotaCienciasNatureza             float64
`json:"notaCienciasNatureza"`
	SituacaoCienciasNatureza         string
`json:"situacaoCienciasNatureza"`
	NotaCienciasHumanas              float64
`json:"notaCienciasHumanas"`
	SituacaoCienciasHumanas          string
`json:"situacaoCienciasHumanas"`
	NotaLinguagens                   float64
`json:"notaLinguagens"`
	SituacaoLinguagens               string
`json:"situacaoLinguagens"`
	NotaMatematica                   float64
`json:"notaMatematica"`
	SituacaoMatematica               string
`json:"situacaoMatematica"`
	NotaRedacao                      float64
`json:"notaRedacao"`
	SituacaoRedacao                  string
`json:"situacaoRedacao"`
}

type RG struct {
   Numero         string `json:"numero"`
   OrgaoExpedidor string `json:"orgaoExpedidor"`
   UF             string `json:"UF"`
}

type Enem struct {
}

func (e *Enem) Init(stub
shim.ChaincodeStubInterface) peer.Response {
```

```
    return shim.Success(nil)
}

func (e *Enem) Invoke(stub
shim.ChaincodeStubInterface) peer.Response {

    fn, args := stub.GetFunctionAndParameters()

    switch fn {

    case "StoreInscricao":
        if len(args) != 1 {
            errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args),
nil)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
        err := e.storeInscricao(stub, args[0])
        if err != nil {
            if chainError, ok :=
err.(customErrors.ChaincodeError); ok {
                return
customErrors.ToPeerResponse(chainError.Info())
            }
            return shim.Error(err.Error())
        }
        return shim.Success(nil)

    case "ReadInscricaoByNumInscricao":
        if len(args) != 1 {
            errorMsg :=
customErrors.NewArgumentLenError(fn, 1, len(args),
nil)
            return
customErrors.ToPeerResponse(errorMsg.Info())
        }
```

```
                            inscricao,    err    :=
e.readInscricaoByNumInscricao(stub, args[0])
       if err != nil {
           return shim.Success(nil)
       }
       if inscricao == nil {
                                     errorMsg    :=
customErrors.NewGenericError("ReadInscricaoByNumIn
scricao", "Nao ha registro de inscricao atrelado o
numero de inscricao fornecido", nil)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
       }
                      inscricaoInBytes,    err   :=
json.Marshal(inscricao)
       if err != nil {
                                     errorMsg    :=
customErrors.NewInvalidStructError(fn,
"Inscricao", err)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       return shim.Success(inscricaoInBytes)

   case "StoreResultado":
       if len(args) != 1 {
                                     errorMsg    :=
customErrors.NewArgumentLenError(fn, 1, len(args),
nil)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       var result Result
           err  :=  json.Unmarshal([]byte(args[0]),
&result)
       if err != nil {
```

```
                                    errorMsg   :=
customErrors.NewMarshallingError(fn, args[0], err)
                                          return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       err = e.storeResultado(stub, args[0])
       if err != nil {
                           if  chainError,  ok  :=
err.(customErrors.ChaincodeError); ok {
                                          return
customErrors.ToPeerResponse(chainError.Info())
           }
                                    errorMsg   :=
customErrors.NewGenericError("StoreResultado",
"Falha ao armazenar o resultado", err)
                                          return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       return shim.Success(nil)

   case "ReadResultadoByNumInscricao":
       if len(args) != 1 {
                                    errorMsg   :=
customErrors.NewArgumentLenError(fn, 1, len(args),
nil)
                                          return
customErrors.ToPeerResponse(errorMsg.Info())
       }
                              result,   err   :=
e.readResultadoByNumInscricao(stub, args[0])
       if err != nil {
           return shim.Success(nil)
       }
       if result == nil {
                                    errorMsg   :=
customErrors.NewGenericError("ReadResultadoByNumIn
```

```
scricao", "Nao ha registro de resultado atrelado
ao numero de inscricao fornecido", nil)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       resultInBytes, err := json.Marshal(result)
       if err != nil {
                                       errorMsg    :=
customErrors.NewInvalidStructError(fn,
"Resultado", nil)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
       }
       return shim.Success(resultInBytes)

   default:
                                      errorMsg     :=
customErrors.NewUndefinedFunctionError(fn, nil)
                                            return
customErrors.ToPeerResponse(errorMsg.Info())
   }
}

func      (e      *Enem)      storeInscricao(stub
shim.ChaincodeStubInterface, args string) error {
   const funcName = "StoreInscricao"

   var inscricao Inscricao
   err := json.Unmarshal([]byte(args), &inscricao)
   if err != nil {
                                      errorMsg     :=
customErrors.NewMarshallingError(funcName,   args,
err)
       return errorMsg
   }
```

```
        inscricaoKey    :=    "INSCRICAO_"    +
inscricao.NumInscricao

            inscricaoExistente,     err     :=
stub.GetState(inscricaoKey)
   if err != nil {
                                  errorMsg     :=
customErrors.NewGenericError(funcName,  "Erro  ao
verificar  a  existencia  do  numero  de  inscricao",
err)
       return errorMsg
   }
   if inscricaoExistente != nil {
                                  errorMsg     :=
customErrors.NewGenericError(funcName,  "Numero  de
inscricao ja tem inscricao atrelada", nil)
       return errorMsg
   }
   inscricaoJSON, err := json.Marshal(inscricao)
   if err != nil {
                                  errorMsg     :=
customErrors.NewMarshallingError(funcName,   args,
err)
       return errorMsg
   }
        err    =    stub.PutState(inscricaoKey,
inscricaoJSON)
   if err != nil {
                                  errorMsg     :=
customErrors.NewUpdateWorldStateError(funcName,
err)
       return errorMsg
   }
   fmt.Println("Inscricao armazenada, sob o numero
de inscricao ", inscricao.NumInscricao)
   return nil
}
```

```
func (e *Enem) readInscricaoByNumInscricao(stub shim.ChaincodeStubInterface, numInscricao string) (*Inscricao, error) {
   const funcName = "ReadInscricaoByNumInscricao"
   inscricaoKey := "INSCRICAO_" + numInscricao

                inscricaoEnemJSON, err := stub.GetState(inscricaoKey)
   if err != nil {
                                    errorMsg := customErrors.NewGenericError(funcName, numInscricao, err)
          fmt.Println("Erro ao obter dados sob o numero de inscricao ", numInscricao, "-", err)
       return nil, errorMsg
   }
   if inscricaoEnemJSON == nil {
        fmt.Println("Não ha inscricao com o numero ", numInscricao)
       return nil, nil
   }
   var inscricao Inscricao
        err = json.Unmarshal(inscricaoEnemJSON, &inscricao)
   if err != nil {
                                    errorMsg := customErrors.NewMarshallingError(funcName, "Inscricao", err)
       return nil, errorMsg
   }
   return &inscricao, nil
}

func (e *Enem) storeResultado(stub shim.ChaincodeStubInterface, args string) error {
   const funcName = "StoreResultado"
```

```
var result Result
err := json.Unmarshal([]byte(args), &result)
if err != nil {
errorMsg :=
customErrors.NewMarshallingError(funcName, args,
nil)
    return errorMsg
}

resultadoKey := "RESULTADO_" +
result.NumInscricao

inscricaoExistente, err :=
e.readInscricaoByNumInscricao(stub,
result.NumInscricao)
if err != nil {
errorMsg :=
customErrors.NewGenericError(funcName, "Erro ao
verificar a existencia da inscricao", err)
    return errorMsg
}
if inscricaoExistente == nil {
errorMsg :=
customErrors.NewGenericError(funcName, "Nao ha
registro de inscricao atrelado ao numero de
inscricao fornecido, logo, nao pode ser cadastrado
um resultado", nil)
    return errorMsg
}
existingResultadoBytes, err :=
stub.GetState(resultadoKey)
if err != nil {
errorMsg :=
customErrors.NewGenericError(funcName, "Erro ao
verificar a existencia do resultado", err)
    return errorMsg
```

```
    }
    if existingResultadoBytes != nil {
                                    errorMsg     :=
customErrors.NewGenericError(funcName, "Ja existe
resultado atrelado ao numero de inscricao
fornecido", nil)
        return errorMsg
    }
    resultJSON, err := json.Marshal(result)
    if err != nil {
                                    errorMsg     :=
customErrors.NewMarshallingError(funcName,   args,
nil)
        return errorMsg
    }
    err = stub.PutState(resultadoKey, resultJSON)
    if err != nil {
                                    errorMsg     :=
customErrors.NewUpdateWorldStateError(funcName,
nil)
        return errorMsg
    }
    return nil
}

func  (e  *Enem)  readResultadoByNumInscricao(stub
shim.ChaincodeStubInterface,  numInscricao  string)
(*Result, error) {
    const funcName = "ReadResultadoByNumInscricao"

    resultadoKey := "RESULTADO_" + numInscricao

    resultJSON, err := stub.GetState(resultadoKey)
    if err != nil {
                                    errorMsg     :=
customErrors.NewGenericError(funcName,
numInscricao, err)
```

```
            fmt.Println("Erro ao obter dados com o
numero de inscricao ", numInscricao, "-", err)
        return nil, errorMsg
    }
    if resultJSON == nil {
         fmt.Println("Não ha resultado com o numero
de inscricao ", numInscricao)
        return nil, nil
    }
    var result Result
    err = json.Unmarshal(resultJSON, &result)
    if err != nil {
                                    errorMsg      :=
customErrors.NewMarshallingError(funcName,
"Result", err)
        return nil, errorMsg
    }
    fmt.Println("Sucesso ao obter o resultado sob o
numero de inscricao ", result.NumInscricao)
    return &result, nil
}

func main() {
    err := shim.Start(new(Enem))
    if err != nil {
            fmt.Printf("Erro iniciando a chaincode:
%s", err)
    }
}
```