PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA POLITÉCNICA E DE ARTES

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

USANDO BLOCKCHAIN PARA EMITIR CERTIFICADOS E DIPLOMAS

VALTECI MARCELINO COELHO JUNIOR

GOIÂNIA

2023

VALTECI MARCELINO COELHO JUNIOR

USANDO BLOCKCHAIN PARA EMITIR CERTIFICADOS E DIPLOMAS

Trabalho de Conclusão de Curso apresentado à escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Daniel Correa da Silva.

GOIÂNIA

2023

VALTECI MARCELINO COELHO JUNIOR

USANDO BLOCKCHAIN PARA EMITIR CERTIFICADOS E DIPLOMAS

Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Ciência da Computação, e aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, em _____/_____/_____.

_____

Profa. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso.

Banca examinadora:

_____

Orientador: Prof. Me. Daniel Correa Da Silva

_____

Membro 1: Prof. Me. Olegário Corrêa da Silva Neto

_____

Membro 2: Prof. Me. Eugênio Júlio M. Candido Carvalho

GOIÂNIA

2023

RESUMO

Este trabalho demonstra uma possível forma de se validar um certificado ou um diploma emitido por uma instituição de ensino, mesmo se ela não existir mais, ou perder esses documentos. O que permitiu viabilizar tudo isso foi a tecnologia *blockchain*, que tem crescido desde a criação do *bitcoin* em 2009. Os casos de uso da tecnologia *blockchain* vão muito além das criptomoedas, pois foram usadas para o controle de vacinas contra a covid-19, bem como para a criação do real digital no Brasil (Drex). Seguindo a linha de raciocínio deste trabalho, o Massachusetts Institute of Technology (MIT), começou a emitir certificados a seus alunos em *blockchain* a partir de 2017. Para demostrar na prática como fazer isso, uma aplicação web que simula o ambiente de uma instituição de ensino foi construída neste trabalho para esse propósito. A aplicação, bem como as instruções para executá-la estão disponíveis neste link: https://github.com/valteci/emissao-certificados-blockchain.

Palavras-chave: B*lockchain*. *Smart contracts. Ethereum*. Certificados e diplomas. Validação de documentos.

ABSTRACT

This paper demonstrates a possible way to validate a certificate or diploma issued by an educational institution, even if it no longer exists or loses these documents. What made all of this feasible was blockchain technology, which has grown since the creation of Bitcoin in 2009. The use cases of blockchain technology go far beyond cryptocurrencies, as they have been used for tracking COVID-19 vaccines, as well as for the creation of Brazil's digital real (Drex). Following the line of reasoning of this paper, the Massachusetts Institute of Technology (MIT) has began issuing certificates to its students on the blockchain as of 2017. To demonstrate practically how to do this, a web application simulating the environment of an educational institution was built in this paper for this purpose. The application, as well as the instructions to run it, are available at this link: https://github.com/valteci/emissao-certificados-blockchain.

Keywords: Blockchain. Smart contracts. Ethereum. Certificates and diplomas. Document validation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATUREAS E SIGLAS

RNDS = Rede Nacional de Dados de Saúde

MIT = Massachusetts Institute of Technology

CEO = Chief Executive Officer

NIST = National Institute of Standards and Technology

CPF = Cadastro de Pessoa Física

POW = Proof of Work

POS = Proof of Stake

ECDSA = Elliptic Curve Digital Signature Algorithm

EVM = Ethereum Virtual Machine

DNS = Domain Name System

ENS = Ethereum name Service

RPC = Remote Procedure Call

JWT = Json Web Token

ORM = Object-Relational Mapping

NFT = Non Fungible Token

SUMÁRIO

1 INTRODUÇÃO

A tecnologia *blockchain* ganhou grande destaque desde que Nakamoto (2008) publicou seu *paper* em 2008 sobre a famosa criptomoeda *bitcoin*, no entanto, as aplicações dessa tecnologia vão muito além do escopo das criptomoedas, abrangendo casos de uso na área da economia, educação, indústria, saúde, governança, além de muitas outras (RODRIGUES, 2019, p. 29).

Blockchain também ganhou destaque porque foi usado no real digital (DREX), desenvolvido pelo banco central, o que facilitou o seu desenvolvimento (MIT Technology Review, 2023), bem como foi usado pela Rede Nacional de Dados de Saúde (RNDS), que alimenta o ConecteSUS, impedindo que doses de vacinas sejam apagadas sem deixar rastros (MORENO et al., 2023).

Nesse contexto, destaca-se o uso de *blockchain* para emissão de certificados por instituições de ensino aos seus alunos. O Massachusetts Institute of Technology (MIT) foi uma das primeiras a emitir diplomas digitais em *blockchain* e ele ressalta que isso dá aos alunos autonomia sobre seu próprio histórico (DURANT, 2017).

O cofundador e *Chief Executive Officer* (CEO) da Learning Machine, Chris Jagers, afirma que “o MIT emitiu registros oficiais em um formato que pode existir mesmo se a instituição desaparecer, mesmo se nós desaparecermos como fornecedores” (DURANT, 2017, tradução nossa). Isso significa que, ao emitir um diploma ou certificado numa *blockchain*, ainda é possível verificar a sua validade, mesmo se a instituição que o emitiu não existir mais ou não tiver o certificado.

Num fluxo normal de interação entre estudante e instituição, o estudante faz a matrícula num curso, completa o curso e então a instituição emite o certificado ao aluno. No caso de o certificado ficar armazenado apenas no banco de dados da instituição, torna-se custoso verificar sua validade uma vez que a instituição que o emitiu não existe mais ou o perdeu.

Para o *National Institute of Standards and Technology* (NIST) (2018), *blockchains* são tecnologias muito boas em fornecer transparência e auditabilidade, além de poderem ser altamente resistentes à censura, o que é desejável num contexto de emissão de certificados ou diplomas.

Introduzindo o conceito de *blockchain* nesse cenário, o aluno e a instituição de ensino teriam ambos um par de chaves, pública e privada e um endereço na *blockchain*.

O Endereço da *blockchain* é um identificador único na *blockchain* derivado da chave privada que foi gerada. Nesse caso, é possível usar o endereço na *blockchain* para fins de autenticidade. Transações na *blockchain* também possuem seus próprios identificadores únicos e o endereço emissor que a originou (NIST, 2020). Este trabalho se apoiou nessas propriedades das *blockchains* para fornecer a solução.

A instituição de ensino, tendo uma chave privada e o respectivo endereço na *blockchain*, poderia emitir o diploma ou certificado na *blockchain* com os dados do aluno. Isso deixa não só o diploma ou certificado armazenado na *blockchain* para sempre, mas também a transação que o originou, desse modo, pode-se verificar essas transações mesmo se a instituição que as criou não existir mais.

Para auferir a validade de um certificado que se encontra numa *blockchain*, é necessário saber o endereço na *blockchain* da instituição de ensino, que deve ser ativamente divulgado por ela. Com isso, pode-se rastrear a transação que gerou o certificado ou diploma e verificar se os dados que nele constam equivalem aos dados pessoais do aluno, por exemplo, nome completo e 6 dígitos do cadastro de pessoa física (CPF).

Diante deste cenário, as instituições de ensino que quiserem adotar tal modelo, não precisam abandonar o modelo de emissão que já usam, ou seja, a emissão de diplomas e certificados em *blockchain* não precisa substituir o modelo que elas já utilizam, servindo apenas como mais uma camada de garantia e proteção para os estudantes.

O capítulo 3 (metodologia) deste trabalho apresenta um exemplo de implementação desse modelo, simulando um ambiente de uma instituição de ensino, numa página web, cobrindo desde a matrícula do aluno até a emissão de fato do certificado na *blockchain* e validação do mesmo. A rede *blockchain* escolhida para a emissão do certificado foi a Ethereum, por sua grande comunidade e base de conhecimento já disponível.

1.1 Objetivos Principal e Específicos

O objetivo principal deste trabalho é construir uma aplicação web para emissão, consulta e validação de diplomas ou certificados utilizando a *blockchain* da *Ethereum*, demostrando os principais conceitos do modelo de emissão de certificados em *blockchain*, e serve como um exemplo no qual instituições de ensino, empresas, desenvolvedores e acadêmicos possam se inspirar.

Este trabalho também tem os seguintes objetivos específicos:

- Levantamento dos requisitos da aplicação web a ser construída.
- Modelagem da aplicação.
- Implementação do *back-end* e *front-end* da aplicação web.
- Integração do *back-end* da aplicação com a rede de teste da *Ethereum*.
- Geração de contas *Ethereum* no *front-end* da aplicação.
- Estudar a viabilidade e escalabilidade da aplicação.

1.2 Estrutura do Trabalho

Este trabalho está organizado 4 capítulos, nessa ordem: referencial teórico, metodologia, resultados e conclusão. O referencial teórico apresenta uma revisão bibliográfica sobre os principais assuntos abordados no trabalho, destrinchando conceitos basilares para entendimento do trabalho como um todo; o capítulo de metodologia apresenta os métodos usados para criar a solução do problema de pesquisa, mostrando e descrevendo as tecnologias usadas; o capítulo dos resultados mostra os resultados obtidos a partir da solução adotada; por fim, o capítulo de conclusão expõe os resultados deste trabalho como um todo e as suas contribuições, bem como o que pode ser feito em trabalhos futuros para aprimorar ou aumentar o escopo deste trabalho.

2 REFERENCIAL TEÓRICO

Nesse capítulo serão expostos os conceitos basilares para uma melhor compreensão do que é a tecnologia *blockchain*, os contratos inteligentes e a cadeia de suprimentos, mostrando as relações que possuem uma com a outra. A estrutura do referencial teórico nessa pesquisa foi dividida em: Trabalhos relacionados, *Blockchain*, Algoritmos de Consenso, Transações, Carteiras, Contratos Inteligentes, *Ethereum* e *Solidity*.

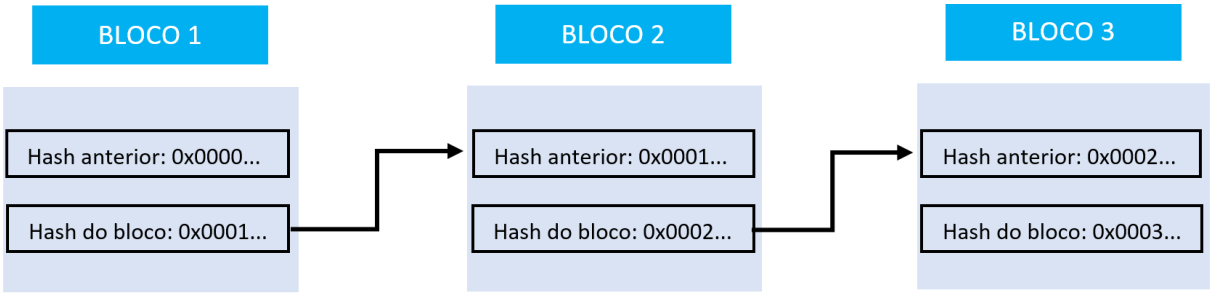
2.1 *Blockchain*

A tecnologia *blockchain* ganhou popularidade devido ao seu uso na rede *bitcoin* em 2009. No entanto, é importante frisar que criptoativos e *blockchain* não são as mesmas coisas, o primeiro apenas faz uso da tecnologia *blockchain* para o seu funcionamento. Já *blockchain* é, na verdade, um banco de dados distribuído, ou *ledger* distribuído, que consiste em uma série de blocos ligados por meio de *hashes* criptográficos (KOK, 2019).

É válido, ainda, afirmar que *blockchains* também podem ser vistas como uma estrutura de dados ou uma rede distribuída, tendo algumas características que as definem, como o fato de serem: descentralizadas, o que significa que esse banco de dados não é mantido por uma única entidade central ou que não depende de intermediários para funcionar; auditáveis e públicas, o que significa que qualquer um pode ver os seus registros e auditá-los; imutável, o que significa que os dados não podem ser alterados depois de registrados; seguras, o que significa que o custo para fazer um ataque bem-sucedido é maior que a recompensa desse ataque (NIST, 2018).

Olhando em alto nível a estrutura típica de uma *blockchain* podemos ver uma cadeia de blocos ligados através de *hashes* criptográficos. Cada um desses blocos tem seu próprio *hash*, que é gerado ao fazer um *hash* dos dados do bloco, além disso, cada bloco guarda dentro de si o *hash* do bloco anterior, o que dá o nome da estrutura “cadeia de blocos” ou *blockchain*. A figura 1 mostra um exemplo resumido de uma estrutura *blockchain*.

Figura *1* - Estrutura básica de uma *blockchain*



Fonte: Elaborado pelo Próprio Autor

2.2 *Hash*

Um *hash* ou uma função *hash* é uma função que recebe uma entrada de dados qualquer e sempre gera uma saída de tamanho fixo para uma determinada entrada, além disso, as funções *hash* devem satisfazer a alguns requisitos para que sejam consideradas seguras e úteis, tais como: (SHARMEELA et al., 2023 p. 271)

1. Unidirecionais: as funções *hash* devem ser unidirecionais, o que significa que dada uma saída qualquer de uma função *hash*, seja intratável calcular a entrada que gerou a saída. Ou, ainda, pode-se dizer que dado y, tal que hash(x) = y, seja intratável computacionalmente obter x a partir de y.

2. Resistentes a colisões: elas devem ser resistentes a colisões, o que significa dizer que deve ser computacionalmente intratável encontrar duas entradas diferentes que geram a mesma saída na função *hash*. Ou, ainda, pode-se dizer que dado duas entradas x e y, seja intratável encontrar valores para x e y, de modo que hash(x) = hash(y).

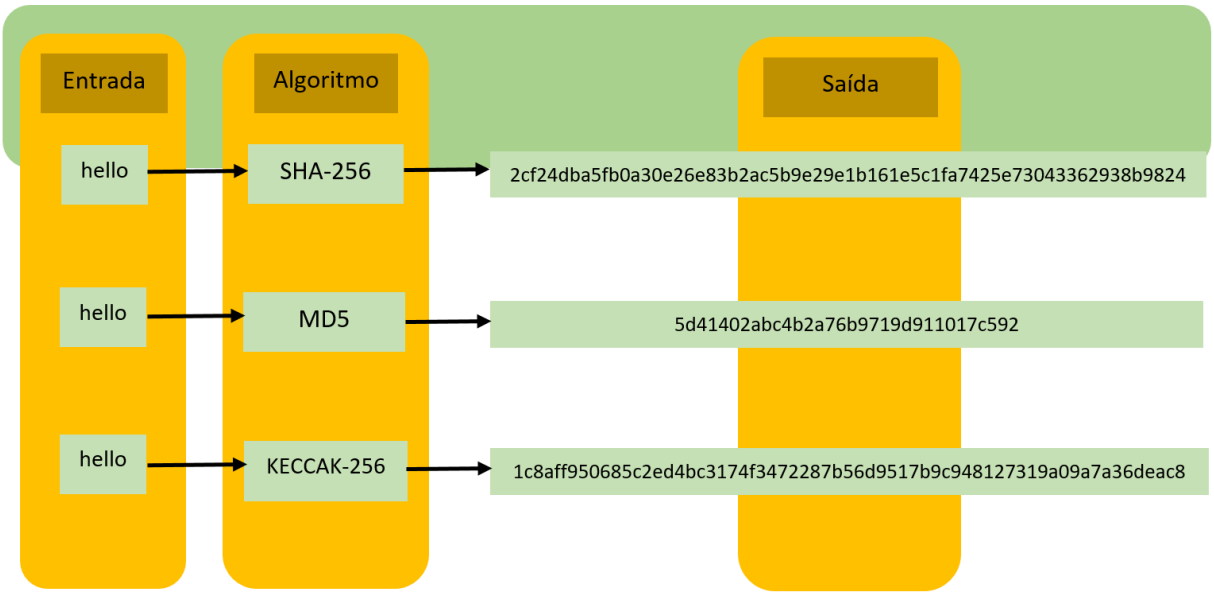
3. Efeito avalanche: elas devem possuir o chamado “efeito avalanche”, o que significa dizer que pequenas alterações na entrada (ainda que seja na ordem de 1 único bit) devem gerar uma grande alteração na saída,

de modo que para quaisquer duas entradas semelhantes, suas saídas não podem ser parecidas.

Existem diversos algoritmos de *hash*, dentre os quais, os mais conhecidos e utilizados até o ano de 2023 são: SHA-256, SHA-512, MD5, Whirlpool, dentre outros. Vale ressaltar que a *blockchain* da *Ethereum* usa o algoritmo de *hash* chamado KECCAK-256 enquanto o *bitcoin* usa o algoritmo de *hash* SHA-256 (KOK, 2019).

Na figura 2 a seguir, são mostrados alguns exemplos usando as funções *hash* com a mesma entrada “hello” e com as suas respectivas saídas.

Figura *2* - Funções *Hash*



Fonte: Elaborado pelo Próprio Autor

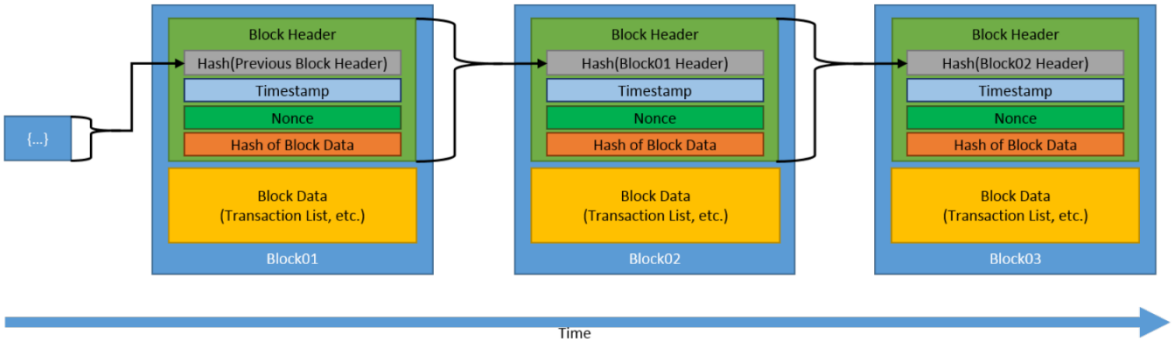
2.3 Bloco

O bloco é uma das partes mais importantes do *blockchain*, e em uma visão de alto nível, pode ser visto como um conjunto de dados ou transações, ou seja, as transações são organizadas em blocos que são posteriormente adicionados ao *blockchain* se ele for validado pela rede (RAWAL *et al.*, 2022).

Os blocos têm um conjunto de campos dentro de si, que são necessários para o funcionamento e organização do *blockchain*, esses campos podem variar de acordo com o tipo de *blockchain* e de acordo com o algoritmo de consenso adotado pela rede *blockchain,* porém há alguns campos que estão presentes na maioria deles (RAWAL *et al.*, 2022).

Geralmente, os blocos são divididos em duas partes: cabeçalho e dados, assim como na figura 3. O cabeçalho contém, normalmente, os metadados sobre os blocos nos campos como *timestamp*, *nonce*, *merkle root*, dentre outros. A parte de dados tem, normalmente, as transações, que podem conter movimentações de valores ou outros tipos de dados (RAWAL *et al.*, 2022).

Figura 3 - Estrutura de um Bloco



Fonte: NIST

2.3.1 Dificuldade

O campo de dificuldade dentro do bloco está presente apenas em *blockchains* que usam o algoritmo de consenso *Proof of Work* (POW) e representa a dificuldade do problema matemático a ser resolvido para criar um bloco na rede. Geralmente, ele define o número de zeros que o *hash* deve começar (Bitcoin wiki, 2021a).

Aumenta-se o número de zeros para aumentar a dificuldade e diminui-se o número de zeros para abaixar a dificuldade. Os detalhes de quando deve ocorrer o ajuste de dificuldade e o nível a ser calibrado pode variar de *blockchain* para blockchain (Bitcoin wiki, 2021a).

2.3.2 *Hash* do Cabeçalho do Bloco Anterior

Cada bloco, exceto o primeiro (bloco gênesis), tem um *hash* do cabeçalho do bloco anterior dentro do seu próprio cabeçalho, como na figura 1. Isso é feito para vincular o bloco atual ao bloco anterior (Bitcoin wiki, 2021b).

Em algoritmos POW esses *hahes* devem começar com uma quantidade específica de zeros (normalmente indicada no campo d*ifficulty*), com isso, ao se alterar apenas um único bit de qualquer bloco anterior, todos os blocos subsequentes vão ter seus *hahes* alterados, não começando mais com a quantidade de zeros determinada, o que faz deles blocos inválidos. Algumas *blockchains* podem fazer um *hash* do bloco inteiro em vez de usar apenas o *hash* do cabeçalho. (Bitcoin wiki, 2021b).

2.3.3 *Hash* dos Dados do Bloco

Cada bloco tem um *hash* da parte dos dados do próprio bloco. Esse campo fica dentro do cabeçalho do bloco, conforme a figura 1, e é usado para garantir a imutabilidade dos dados dentro de cada bloco na *blockchain*. Como esse campo está dentro do cabeçalho, e o *hash* do cabeçalho está dentro do cabeçalho do bloco seguinte, então qualquer alteração na parte dos dados gera invalidação dos blocos subsequentes (Bitcoin wiki, 2021b).

2.3.4 *Nonce*

Presente apenas em *blockchains* que usam o algoritmo de consenso POW, *nonce* é número que junto com o *hash* do cabeçalho do bloco (ou *hash* do bloco em algumas implementações) gera um *hash* que começa com uma certa quantidade de zeros que, normalmente, é definida pelo campo *difficult*. Assim, ao unir o *hash* do cabeçalho do bloco com o *nonce* e fazer o *hash* dessa união, o resultado gerado tem que ser igual a uma saída que começa com uma determinada quantidade de zeros (Bitcoin wiki, 2021b).

Para montar um bloco válido, mineradores, normalmente, variam o número do *nonce* até encontrar um *hash* que atenda aos critérios de dificuldade estabelecidos pela rede (Bitcoin wiki, 2021b).

2.3.5 Raiz Markle (*Markle root*)

A *Markle root é* a raiz da *merkle tree* (árvore *merkle*). A *merkle tree* é uma estrutura de dados do tipo árvore binária que tem como folhas o *hash* de cada

transação presente no bloco, o nó pai dessas folhas é o *hash* dos *hashes* concatenados dos filhos, ou seja, concatena-se o *hash* dos filhos e extrai-se um novo *hash* em cima disso, que será o *hash* do nó pai. Esse processo se repete até que haja apenas um único *hash*, chamado de *merkle root*. Essa estrutura é muito útil, pois, com o uso dela, são preciso apenas 2*log(N) operações no pior caso, sendo N o número de transações (NIST, 2018).

2.3.6 Versão

Esse campo indica qual é a versão do protocolo ou software usado no momento da publicação do bloco na *blockchain*. Esse campo identifica qual a versão do software ou protocolo estava sendo usado na época de publicação do bloco. Ao auferir a validade de um bloco, deve-se considerar as regras que estavam sendo usadas na versão especificada por esse campo e não as da versão atual (Bitcoin Wiki, 2021c).

2.3.7 Carimbo de Data e Hora (*Timestamp*)

O *timestamp* (carimbo de data e hora) representa a data, incluindo horas, minutos e segundos, de publicação do bloco. Várias *blockchain*, incluindo o *bitcoin*, usam um modelo chamado *epoch unix* que contabiliza os segundos decorridos desde 00:00:00 UTC de 1 de janeiro de 1970 até a data atual. Na prática, esse é um campo de 4 *bytes* que possui um valor numérico, representando a quantidade em segundos desde a data mencionada até a data de publicação do bloco. (Bitcoin Wiki, 2021b).

2.4 Transações

As transações são o principal meio de se interagir com uma *blockchain*, na maioria das *blockchains*, as transações são feitas pelos usuários, que assinam a transação com a respectiva chave privada e a submetem à rede, além disso toda transação tem o endereço do remetente (*from*) e do destinatário (*to*) (SHARMEELA et al., 2023 p. 272).

Várias *blockchains*, tais como a do *Ethereum* e a do *bitcoin*, dão suporte ao *multisign,* que é um recurso que permite que uma transação seja assinada por

diferentes chaves privadas, podendo se exigir uma quantidade mínima de assinatura para que a transação seja efetuada (PRUSTY, 2018, p. 193).

As transações são encaminhadas à rede, cujos nós têm o papel de verificar se a transação é válida (está de acordo com as regras da *blockchain*), de inclui-la em um bloco na *blockchain*, e de propagar o bloco contendo a transação pela rede, quando o bloco estiver pronto (KUANAR *et al.*, 2022, p. 259).

Uma *blockchain* pode ter vários tipos de transações, geralmente, as mais comuns incluem, mas não se limitam a: transferência de valores, travamento de criptomoedas e *tokens* em redes *proof of stake* (prova de participação – POS), registro de dados, implantação e execução de contratos inteligentes (PRUSTY, 2018, p. 50).

Além disso, as *blockchains* costumam usar uma linguagem de script própria que permitem criar as transações e estrutura-las, de acordo com as necessidades do remetente (PRUSTY, 2018, p. 50).

2.5 Algoritmos de Consenso

Toda *blockchain* tem um algoritmo de consenso, que é um elemento fundamental, já que ele influencia na segurança, na descentralização, na velocidade e na escalabilidade da *blockchain*. O consenso se refere a um acordo alcançado entre os nós em relação ao estado da rede (GRINCALAITIS, 2019, p. 34).

Como as *blockchain* públicas não possuem uma entidade central, é por meio do consenso que a rede *blockchain* vai decidir o que permitido o que não é, o que é válido ou inválido e quais ações devem ser penalizadas, ou seja, o consenso define as regras da *blockchain* que os nós seguem e chega a um acordo sobre o estado da rede. Por isso muitas *blockchains* adoram um algoritmo de consenso (GRINCALAITIS, 2019, p. 34).

Existem diversos tipos de algoritmos de consenso, cada um com suas vantagens e desvantagens, no entanto dois principais algoritmos se destacam dentre os demais e são amplamente usados em diversas *blockchain*, trata-se dos algoritmos de consenso de prova de trabalho (em inglês, proof of work, POW) e de prova de participação (em inglês, proof of stake, POS), além disso existem outras variações do algoritmo de consenso POS. (GRINCALAITIS, 2019, p. 34).

2.5.1 Prova de Trabalho

O algoritmo POW foi o primeiro algoritmo de consenso construído em redes *blockchain* e foi implementado no *bitcoin* desde a sua primeira versão em 2009, devido a isso, muitas outras redes *blockchains* adotaram o POW desde então (SHARMEELA et al., 2023).

O POW é um algoritmo de consenso que exige que os nós resolvam um problema matemático, que seja difícil de resolver e fácil de se verificar, para que possam adicionar um novo bloco a *blockchain*, esse processo geralmente exige hardware especializado e alto consumo de energia (NIST, 2018).

Nesse algoritmo, os nós devem montar um bloco de acordo com as regras do protocolo da *blockchain*, de modo que seu *hash* comece com uma certa quantidade de zeros previamente definida. Nesse sistema, os nós competem entre si para tentarem montar um bloco válido, ao montá-lo, esse nó deve transmiti-lo para outros nós (XIAO, 2019, p. 376).

No POW, quando dois ou mais nós conseguem montar um bloco ao mesmo tempo, geralmente, o bloco que chegou primeiro é aceito e o outro é descartado; se eles chegarem ao mesmo tempo, a *blockchain* se bifurca em 2 *blockchains* e espera o próximo bloco para decidir qual das 2 *blockchains* é a certa, caso, novamente, 2 blocos cheguem ao mesmo tempo, o processo se repete até que uma das *blockchains* seja a maior. A figura 4 ilustra esse processo. (NIST, 2018)

2.5.2 Prova de Participação

O algoritmo de prova de participação (em inglês, *proof of stake*, POS) não depende de processamento intensivo ou resolução de um problema matemático, como ocorre no POW, em vez disso, os nós precisam travar *tokens* da rede, numa operação que se chama *stake*, ou seja, os nós compram a criptomoeda da rede e fazem uma transação que as deixa inutilizáveis, até que uma transação chamada *unstake*, que as retira dessa situação, seja feita (SHARMEELA et al., 2023).

Nesse sistema, cada nó tem uma quantidade de *tokens* em *stake*, que é a sua participação, um algoritmo randômico escolhe aleatoriamente qual deles publicará o próximo bloco, porém a distribuição da probabilidade de cada nó ser escolhido está em função da sua quantidade de *tokens* em *stake*. Assim, se um nó tem 10% dos *tokens* em *stake* ele tende a ser escolhido 10% das vezes; se tem 50% dos *tokens* em *stake*, tende a ser escolhido 50% das vezes, e assim por diante. O nó escolhido é responsável por montar o bloco e publicá-lo, sendo recompensado por isso (NIST, 2018, p. 32).

O POS também conta com um mecanismo de punição dos nós que se comportarem mal chamado de *slashing*, que pune o nó por mal comportamento ou inatividade, os tipos de penalizações a serem aplicadas podem variar de *blockchain* para *blockchain*, mas geralmente estão relacionadas com a diminuição de recompensas ou até mesmo com a perda de uma porcentagem do valor em *stake* (RAWAL et al., 2022, p. 100).

Dadas todas essas características do POS, ele consegue ter maior eficiência energética e maior velocidade no processamento de transações se comparado com POW, porém ele tende a deixar a rede menos descentralizada do que o POW (NIST, 2018).

Há variações do POS, sendo a mais conhecida e implementada a prova de participação delegada (em inglês, *delegated proof of stake*, DPOS). No DPOS, os detentores de *tokens* podem delegá-los aos validadores para que estes façam o *stake*, enquanto no POS isso não é possível; outro ponto é que geralmente uma rede DPOS tem uma quantidade limitada de nós, enquanto nas redes POS não há essa restrição (SHARMEELA et al., 2023).

2.6 Contratos Inteligentes

O conceito de contratos inteligentes foi introduzido pela primeira vez por Nick Szabo em 1994 que o descreveu como sendo “um protocolo de transação computadorizada que executa os termos de um contrato” (NIST, 2018).

Os contratos inteligentes são um conjunto de código e dados previamente programados que executa os termos do contrato autonomamente quando todas as

condições no contrato são atendidas, podendo alterar seu estado, o de outros contratos, ou até o da *blockchain*. Eles são implementados na *blockchain* por uma linguagem fornecida por ela, e são executados e armazenados pelos nós da *blockchain*. (NIST, 2018);

Upadrista (2021, p. 176) ainda traz uma definição de alto nível de contrato inteligente como sendo um contrato autoexecutável escrito em linhas de código que está distribuído pelos nós da rede *blockchain*.

Os contratos inteligentes ampliaram as possibilidades da tecnologia *blockchain*, trazendo funções como finanças descentralizadas, *tokens* não fungíveis (NFT), aplicativos descentralizados. Com isso, permite-se executar na *blockchain* aplicações, que aproveitam das vantagens da *blockchain* como descentralização, segurança, imutabilidade e resistência à censura, o que é desejável em algumas aplicações (KOK, 2019, p. 47).

2.7 *Ethereum*

O e*thereum* tem sido a principal *blockchain* de contratos inteligentes desde o seu lançamento em 2015. Seu criador, Vitalik Buterin, a definiu como sendo “Ethereum is the world computer”, ou em português, “Ethereum é o computador mundial” (GRINCALAITIS, 2019).

Ethereum foi lançada originalmente com o algoritmo de consenso POW, porém em setembro de 2022 mudou o algoritmo para POS, numa atualização que ficou conhecida como “*the merge*”, isso possibilitou aumentar as transações por segundo e abaixar os custos das transações (ETHEREUM, 2022).

Essa *blockchain* pode ser vista como um grande computador descentralizado, sendo capaz de executar códigos de programação programados e enviados para rede através de transações (GRINCALAITIS, 2019).

2.8 Contas *Ethereum* e Endereço *Ethereum*

Chama-se “conta *Ethereum*” o conjunto de chaves pública e privada válidos na rede *Ethereum*. O endereço é derivado da chave pública e é usado como meio de identificar a conta Ethereum na rede (NIST, 2020).

Para gerar uma conta *Ethereum*, primeiro é preciso gerar a chave privada que é um número de 32 bytes, basicamente, pode-se apenas gerar 32 bytes de maneira aleatória. Depois extrai-se a chave pública aplicando o *Elliptic Curve Digital Signature Algorithm* (ECDSA), então gera-se um *hash keccak256* da chave pública eliminando o primeiro *byte* da chave pública (que é a versão da chave). Por fim, os 20 últimos bytes desse *hash* será o endereço *Ethereum* da conta (FreeCodeCamp, 2018).

Um código escrito em NodeJS que gera contas *Ethereum* pode ser visto no apêndice A.

2.9 *Solidity*

A maioria das *blockchains*, inclusive a do *bitcoin*, tem uma linguagem ou script que é usado para interagir com a *blockchain*, permitindo transações mais complexas ou a definição de contratos inteligentes (GRINCALAITIS, 2019).

A *Ethereum* usa uma linguagem de programação chamada *solidity* para programar seus contratos inteligentes. O código fonte de alto nível precisa ser compilado para um *bytecode* que a *Ethereum Virtual Machine* (EVM) consiga executar (GRINCALAITIS, 2019).

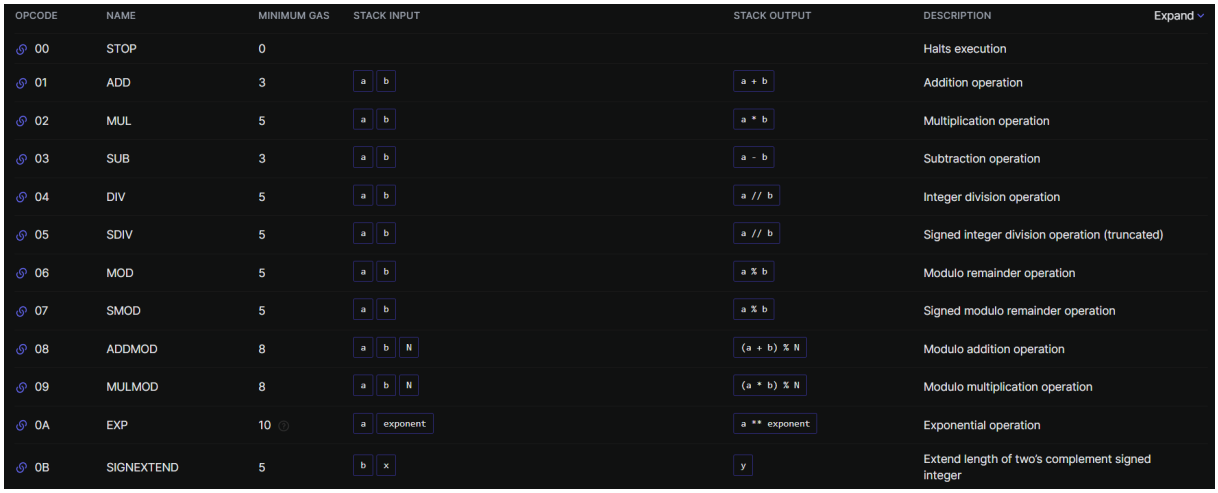
Essa é uma linguagem estaticamente tipada criada por gavin wood, cofundador da *Ethereum* e tem sintaxe parecida com a do javascript. Devido à alta popularidade do *solidity*, outras *blockchains* também adotam ou são compatíveis com ela, permitindo a migração do código mais facilmente (GRINCALAITIS, 2019).

Vale ressaltar que a EVM tem sua própria linguagem de montagem e o *bytecode* gerado pelo código *solidity* representa *opcodes* das instruções, tal como *PUSH* ou *STORE*. Além disso, outro detalhe importante, é que os endereços referenciados nas instruções da linguagem de montagem são, na verdade, endereços *Ethereum* e não endereços de memória (KOK, 2019).

Cada instrução na linguagem de montagem da EVM possui uma taxa associada a ela. Essa taxa chamada de *gas* é medida em *gwei*. 1 *gwei* equivale a 0,000000001 *ethereum*. O preço do *gas* pode variar dependendo da demanda do mercado na rede *Ethereum* (KOK, 2019). A figura 4 mostra o custo mínimo em *gas* de algumas instruções da EVM.

Com isso o custo total para implantar um contrato inteligente na rede é chamado de *gas* e varia a depender de quais e quantas instruções de máquina serão usadas no *bytecode* do contrato inteligente. Por exemplo, o valor da instrução ADD no *bytecode* custa no mínimo 3 *gas* (KOK, 2019).

Figura 4 - Algumas Instruções da EVM



OPCODE	NAME	MINIMUM GAS	STACK INPUT	STACK OUTPUT	DESCRIPTION Expand
00	STOP	0			Halts execution
01	ADD	3	a b	a + b	Addition operation
02	MUL	5	a b	a * b	Multiplication operation
03	SUB	3	a b	a - b	Subtraction operation
04	DIV	5	a b	a // b	Integer division operation
05	SDIV	5	a b	a // b	Signed integer division operation (truncated)
06	MOD	5	a b	a % b	Modulo remainder operation
07	SMOD	5	a b	a % b	Signed modulo remainder operation
08	ADDMOD	8	a b N	(a + b) % N	Modulo addition operation
09	MULMOD	8	a b N	(a * b) % N	Modulo multiplication operation
0A	EXP	10	a exponent	a ** exponent	Exponential operation
0B	SIGNEXTEND	5	b x	y	Extend length of two's complement signed integer

Fonte: evm.codes

2.10 Utilizando *Blockchain* para emissão de documentos

Contratos inteligentes podem ser usados para representar acordos ou documentos financeiros bem como outros tipos de documentos jurídicos genéricos. Além disso, eles também podem representar credenciais de usuários e documentos de identificação (NIST, 2021, p. 52).

Algo muito semelhante ao *Domain Name System* (DNS), existe, na *Ethereum*, o *Ethereum name Service* (ENS). O seu objetivo é mapear um endereço *Ethereum*, que é uma sequência hexadecimal de 20 bytes, para algo que seja legível aos humanos, ou seja, um nome (NIST, 2021, p. 53).

O endereço *Ethereum* é capaz de identificar unicamente uma organização dentro da *blockchain.* Portanto, é possível garantir a autenticidade e o não repúdio de uma transação através do endereço *Ethereum* que emitiu a transação, uma vez que seja sabido a priori o endereço da organização ou seu ENS na *Ethereum.* O que pode ser muito útil na emissão de certificado e diplomas (NIST, 2020).

3 METODOLOGIA

Neste capítulo será apresentada a construção de uma aplicação web que foi construída para simular o ambiente de uma instituição de ensino, demostrando funcionalidades que são desejáveis em um sistema que pretenda emitir certificados ou diplomas em *blockchain*.

A aplicação web desenvolvida servirá como um exemplo que demonstre os principais conceitos vistos no capítulo 2, bem como um exemplo concreto para o tema principal deste trabalho.

3.1 Objetivos da Aplicação

Os principais objetivos dessa aplicação web são:

- Criar contratos inteligentes na rede de testes da *Ethereum, sepolia*. Este contrato representa o certificado ou diploma emitido pela instituição de ensino e contém as informações sobre o aluno e do certificado.
- Integrar o *back-end* da aplicação à rede de testes da *Ethereum*.
- Gerar um endereço *Ethereum* para o aluno.
- Consultar diplomas já emitidos na *blockchain* da *Ethereum*, verificando sua validade e autenticidade.
- Criar opções de criação, leitura, alteração e exclusão, para cursos, turmas e alunos, para simular um ambiente de uma instituição de ensino.

3.2 *Blockchain* Escolhida

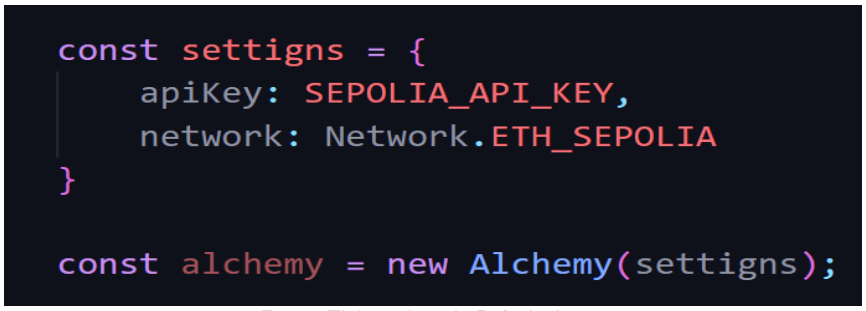
A *Sepolia Testnet*, a principal rede de teste da *Ethereum*, foi escolhida para ser a *blockchain* da aplicação.

A rede de teste é usada por desenvolvedores para testar os códigos dos contratos inteligentes ou protocolos antes de serem implantados na rede principal (*Mainnet*). Isso é análogo a servidores de produção e servidores de teste (ETHEREUM, 2023).

A rede de teste ainda tem a vantagem de não possuir valor monetário, já que uma moeda fictícia é usada para pagar as taxas das transações, o que é desejável em um ambiente de testes.

Na aplicação desenvolvida, como foi utilizado o Alchemy SDK, o que determina se o código está na *mainnet* ou na *testnet* é apenas uma variável nas configurações e a chave da API que é gerada pela Achemy para cada rede. Assim, para fazer que a lógica da aplicação (desde a conexão até as transações) ocorra na *Sepolia Testnet*, basta utilizar as linhas de código que estão na figura 5.

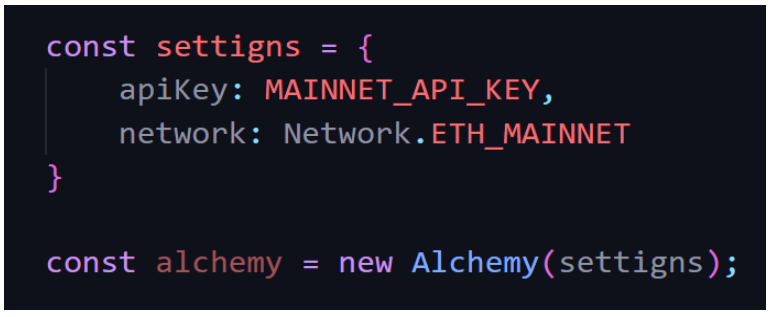
Figura 5 - Configurações da *Sepolia Testnet*



Fonte: Elaborado pelo Próprio Autor

Por outro lado, para fazer que a lógica da aplicação aconteça na *mainnet*, basta mudar os parâmetros do objeto “settigns”, como mostra a figura 6.

Figura 6 - Configurações da *mainnet*



Fonte: Elaborado pelo Próprio Autor

Outra alteração a ser feita se refere ao arquivo hardhat.config, o valor da propriedade defaultNetwork deverá ser “sepolia” para implantar o contrato na rede de teste e “mainnet” para implantar na rede real.

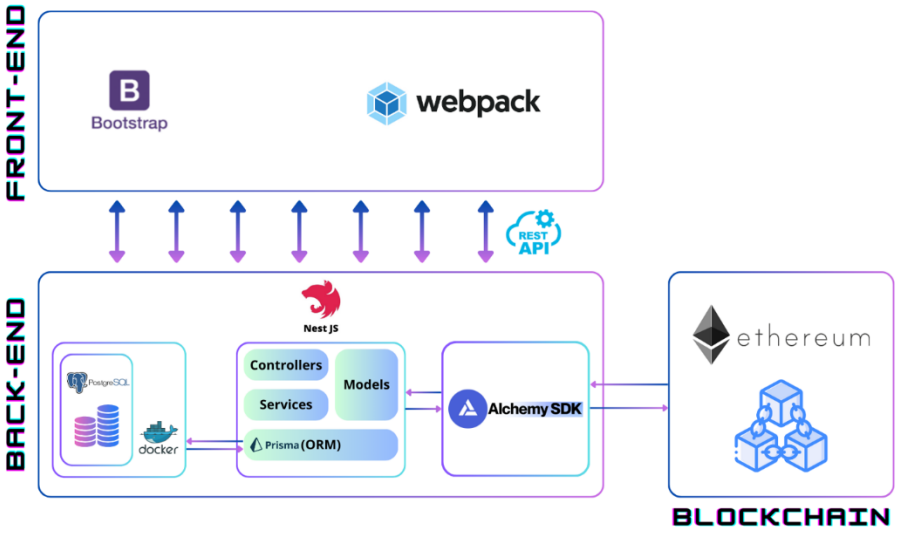
3.3 Arquitetura da Aplicação

A aplicação se divide em 3 grandes partes: *front-end, back-end* e *blockchain*. Conforme a figura 7.

O *front-end* é responsável por lidar com as interações do usuário, fornecendo uma interface gráfica suficiente para coletar os dados necessários. O bootstrap foi usado para construção das páginas web, já o webpack foi responsável por empacotar algumas bibliotecas node.js e disponibilizá-las no ambiente de navegador.

O *back-end* é o servidor da aplicação e é responsável por processar os dados recebidos do *front-end*, bem como retornar respostas que serão apresentadas no *front-end* para o usuário. Foi utilizado o *framework* NestJs para construção do *back-end*, ele divide todo o código em 3 estruturas: *controllers, services e models*. O *controller* é responsável por lidar com requisições que chegam do *front-end*, desde da parte de rotas até a autenticação; os *services* é o local em que fica a lógica de negócio; os *models* lidam com o gerenciamento de dependências, injeção de dependência, permitindo reutilizar código já feito em outros locais. O banco de dados foi executado num container docker e se utilizou o prisma para acessá-lo. Por fim, a biblioteca Alchemy foi usada para conectar o *back-end* à *blockchain Ethereum*.

Figura 7 - Arquitetura da Aplicação



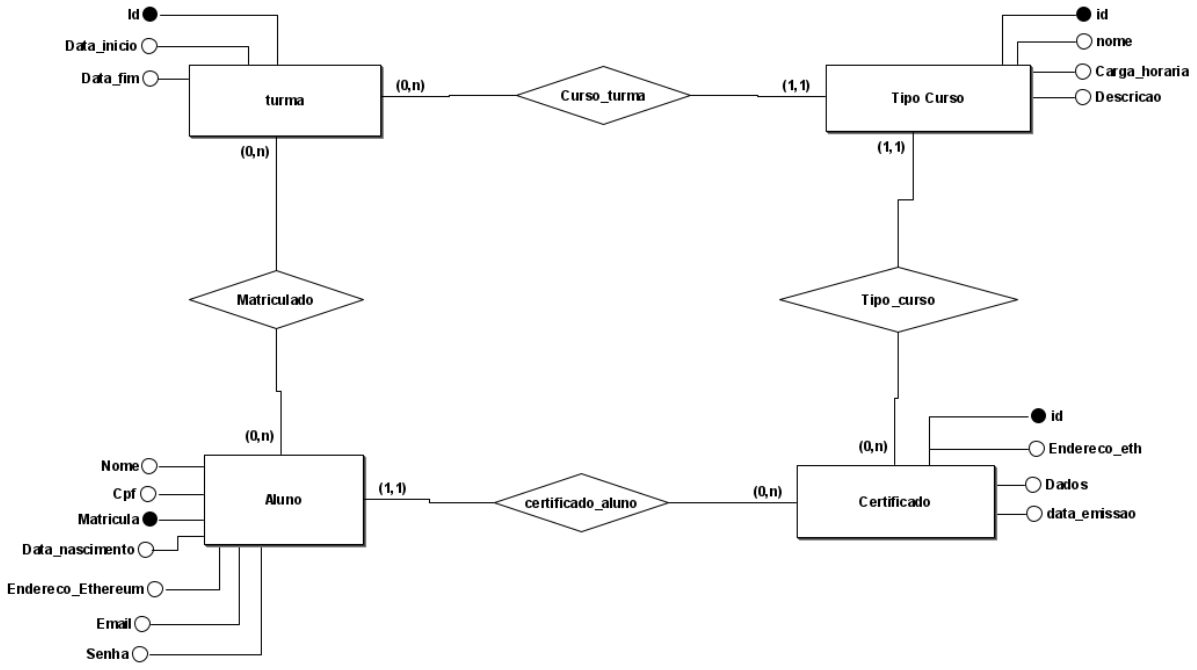
Fonte: Elaborado pelo Próprio Autor

Por fim, a *blockchain Ethereum*, é apenas um nó *Ethereum* à qual o *back-end* se conecta. Nesse caso, o nó *Ethereum* (que contém a cópia sincronizada da *blockchain*) é a Alchemy, não precisando fazer um *Remote Procedure Call* (RPC) ou *WebSocket* para se conectar a um nó *Ethereum*, tudo isso é feito pela Alchemy, através de sua API.

3.4 Modelo Entidade Relacionamento da Aplicação

O modelo entidade relacionamento da aplicação está definido na figura 8. O modelo possui 4 entidades e 4 relacionamentos.

Figura 8 - Modelo Entidade Relacionamento Da Aplicação



Fonte: Elaborado pelo Próprio Autor

A entidade aluno representa um aluno que se matricula na instituição de ensino, atributos dessa entidade representam tipicamente os dados coletados para fazer uma matrícula em algum curso, destacando-se, porém, o atributo “Endereco_Ethereum”, que representa um endereço *Ethereum* que o aluno tenha e que pode ser usado como um identificador único dentro da *Ethereum*, conforme fala NIST (2021). A Matrícula do aluno é a junção do seu CPF com ano em que a matrícula é feita.

A entidade certificado representa um certificado ou diploma que é emitido para o aluno. Esse certificado possui 4 atributos: id, endereco_eth, dados e data_emissao. Como ele pode ser emitido na *blockchain Ethereum* ele tem um endereço *Ethereum* vinculado a ele, o qual pode identificá-lo de forma única, porém como a instituição pode optar em não emitir o certificado na *blockchain*, esse atributo não é chave primária, já que não é obrigatório, ou seja, a instituição pode emitir o certificado apenas em seu próprio banco de dados, se desejar.

A entidade curso representa um tipo de curso oferecido pela instituição, dados como nome do curso, carga horária do curso e sua descrição estão presentes nos atributos dessa entidade.

A entidade turma representa uma turma concreta de algum curso que está sendo oferecido ou que já foi oferecido pela instituição de ensino.

O relacionamento Certificado_aluno vincula cada certificado a algum aluno. Um aluno pode ter vários certificados, mas um certificado só pode estar vincula a um aluno.

O relacionamento Tipo_Curso vincula um tipo de curso ao certificado. Todo certificado deve ter 1 e apenas 1 tipo de curso, enquanto um tipo de curso pode ou não estar vinculado a vários certificados.

Como cada turma tem quer tem que lecionar obrigatoriamente algum curso, o relacionamento Curso_turma vincula cada turma a algum tipo de curso. Os tipos de curso podem estar vinculados a várias turmas, mas uma turma só pode estar vinculada a um tipo de curso.

Por fim, o relacionamento Matriculado garante que uma turma possa ter vários alunos e um aluno possa estar em várias turmas.

3.5 Funcionalidades da aplicação

Basicamente as funcionalidades da aplicação consistem em operação de criação, consulta, alteração e remoção das entidades do diagrama de entidade relacionamento da figura 8, bem como as funcionalidades da tabela 1.

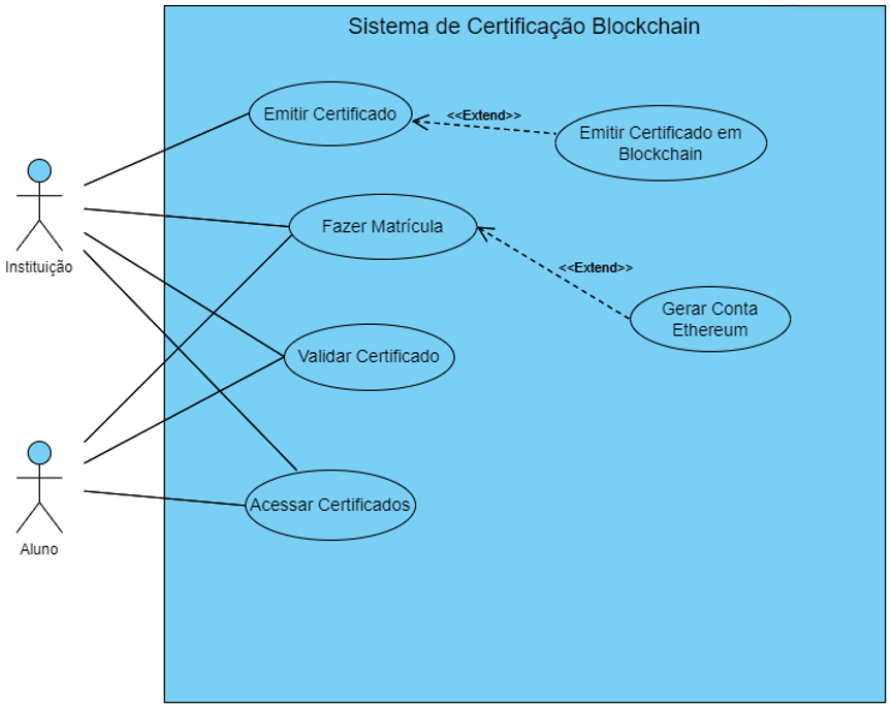
Tabela 1 - Funcionalidades da Aplicação

Funcionalidade	Descrição
Matricular Aluno	A aplicação deve fornecer a capacidade de o próprio aluno se matricular no sistema da instituição. Qualquer pessoa pode se matricular.
Alterar Aluno	Alteração dos dados cadastrais do aluno. O aluno pode alterar seus próprios dados, e a instituição pode alterar dados de qualquer aluno.
Emitir Certificado Em *Blockchain*	Na hora de se cadastrar um certificado no sistema da instituição, tem de haver uma opção que permita emitir, também, o certificado na *blockchain*.
Validar Certificado	Dado um endereço *Ethereum* de um certificado. É preciso verificar se ele é válido, consultando a *blockchain*.
Acessar certificados	O aluno deve ser capaz de acessar os seus certificados.
Baixar certificados	O aluno deve ser capaz de baixar cada um de seus certificados em formato PDF.
Gerar conta *Ethereum*	Na hora da matrícula, deve ser dada a opção ao aluno de se gerar uma conta *Ethereum*.

Fonte: Elaborado pelo Próprio Autor

Com base nas funcionalidades descritas, é possível chegar no diagrama de casos de uso da figura 9.

Figura 9 - Diagrama de Casos de Uso



Fonte: Elaborado pelo Próprio Autor

3.6 Restrições da Aplicação

As restrições da aplicação são:

- **Plataforma:** A aplicação deve funcionar em ambiente web e de forma distribuída.
- **Responsividade:** A aplicação deve funcionar adequadamente em dispositivos de diferentes tamanhos de tela.
- **Autenticação:** Todas as funcionalidades que podem ser executadas apenas pela instituição precisão de autenticação. Todas as funcionalidades que podem ser feitas pelo aluno precisão de autenticação. As funcionalidades de validar certificado, matricular-se e gerar endereço *Ethereum* não precisão de autenticação. A autenticação deve ser feita via e-mail e senha, usando a tecnologia *Json Web Token* (JWT). Os JWTs devem expirar em 15 minutos.

3.7 Recursos Utilizados

Os recursos tecnológicos utilizados para a construção dessa aplicação, o propósito pelos quais foram usados e a respectiva referência ao site oficial do recurso estão presentes na tabela 2.

Tabela 2 - Recursos Utilizados

Recurso	Descrição
Nest JS [1]	NestJs é um framework para construir aplicações do lado do servidor de forma eficiente. Ele foi usado para construir o *back-end* da aplicação.
docker [2]	O docker consegue empacotar e executar um aplicativo em um ambiente isolado chamado contêiner. Ele foi usado para executar e configurar o banco de dados da aplicação.
PostgreSQL [3]	O PostgreSQL é um banco de dados relacional. Ele foi o banco de dados da aplicação.
Prisma [4]	O Prisma se define como um *Object-Relational Mapping* (ORM) da próxima geração. Ele foi responsável por facilitar a interação com o banco de dados na aplicação.
 ethereum-cryptography [5]	ethereum-cryptography é uma biblioteca node.js de operações criptográficas da *Blockchain Ethereum*. Ela foi usada para gerar endereços *Ethereum* para os alunos.
webpack [6]	O webpack é um empacotador (*bundler*) de módulos estáticos do javascript. Ele foi usado para disponibilizar a biblioteca ethereum-cryptography no navegador web em que o *front-end* foi executado.

[1] https://nestjs.com/
[2] https://www.docker.com/
[3] https://www.postgresql.org/
[4] https://www.prisma.io/
[5] https://www.npmjs.com/package/ethereum-cryptography
[6] https://webpack.js.org/

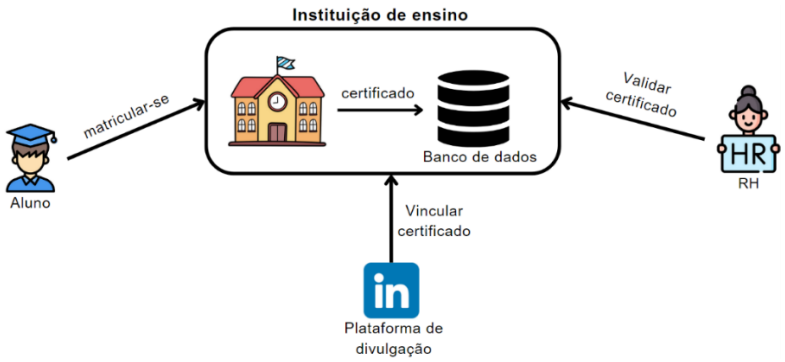
Alchemy SDK [7]	O AlchemySDK é um conjunto de bibliotecas construídas pela Achemy para facilitar a interação com várias *blockchains*. Foi usado para interagir com a *blockchain Ethereum*.
Hardhat [8]	Hardhat é um ambiente de desenvolvimento *Ethereum*. Foi usado para compilar e implantar o certificado construído em forma de contrato inteligente, além de publicá-lo no explorador de blocos.
SOLIDITY [9]	Solidity é a linguagem de programação usada para programar contratos inteligentes na *Ethereum*.
Ethers [10]	Ethers é uma biblioteca javascript usada pelo hardhat para interagir com a *blockchain Ethereum*.
Etherscan [11]	O Etherscan é um explorador de blocos da *Ethereum*, é possível ver todas as transações da *blockchain* nele em tempo real. Ele foi usado para publicar e ver contrato inteligente emitido pela aplicação.
Bootstrap [12]	O Bootstrap se define como uma caixa de ferramentas prontas para construção de front-end. Ele foi suado para construir o *front-end* da aplicação.
Sepolia Testnet [13]	A Sepolia Testnet é a principal rede de testes da *Ethereum*. Ela foi a *blockchain* usada pela aplicação para emitir os contratos inteligentes.

[7] https://www.alchemy.com/
[8] https://hardhat.org/
[9] https://soliditylang.org/
[10] https://docs.ethers.org/v5/
[11] https://etherscan.io/
[12] https://getbootstrap.com/
[13] https://sepolia.dev/

3.8 Como validar o certificado na *blockchain*

Um certificado ou diploma que é emitido apenas no banco de dados da instituição pode ter sua validade conferida, ao acessar esse mesmo banco de dados e conferir se informações no certificado ou diploma realmente são iguais àquelas que estão no banco de dados da empresa. Nesse caso, plataformas de divulgação, funcionários de recursos humanos (RH) e o próprio aluno dependem da instituição que emitiu esse certificado para validá-lo, como mostra a figura 10

Figura 10 - Emissão de Certificados de forma tradicional



Fonte: Elaborado pelo Próprio Autor

Supondo que a instituição tenha uma conta *Ethereum* (uma chave privada e um endereço). Agora, também é possível emitir o certificado na *blockchain*. Uma maneira de fazer isso é criar um contrato inteligente na *Ethereum* e inserir os dados do aluno e do certificado dentro desse contrato. O contrato inteligente que a aplicação usou e que representou o certificado do aluno na *blockchain* está na figura 11.

Quando esse contrato for implantando na *blockchain*, será possível ver o endereço que o implantou, como foi a instituição que emitiu o certificado, o endereço que aparece é o da instituição. Logo, para verificar a autenticidade de um contrato, basta verificar se o endereço que implantou o contrato coincide com o endereço da instituição de ensino, se forem iguais o certificado foi emitido pela instituição, se não, ele não foi emitido pela instituição.

Figura 11 - Contrato Inteligente, representando o certificado

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.7.3;

contract Certificado {

    address public endereco_estudante;
    string public nome;
    string public cpf;
    string public nome_do_curso;
    string public descricao_curso;

    constructor(
        string memory _nome,
        string memory _cpf,
        string memory _nome_do_curso,
        string memory _descricao_curso,
        address enderedo_do_estudante
        ) {

        endereco_estudante = enderedo_do_estudante;
        nome = _nome;
        cpf = _cpf;
        nome_do_curso = _nome_do_curso;
        descricao_curso = _descricao_curso;
    }
}
```

Fonte: Elaborado pelo Próprio Autor

4 RESULTADOS

Nesse capítulo, serão apresentados os resultados obtidos com a criação da aplicação definida no capítulo 3.

4.1 *Front-end*

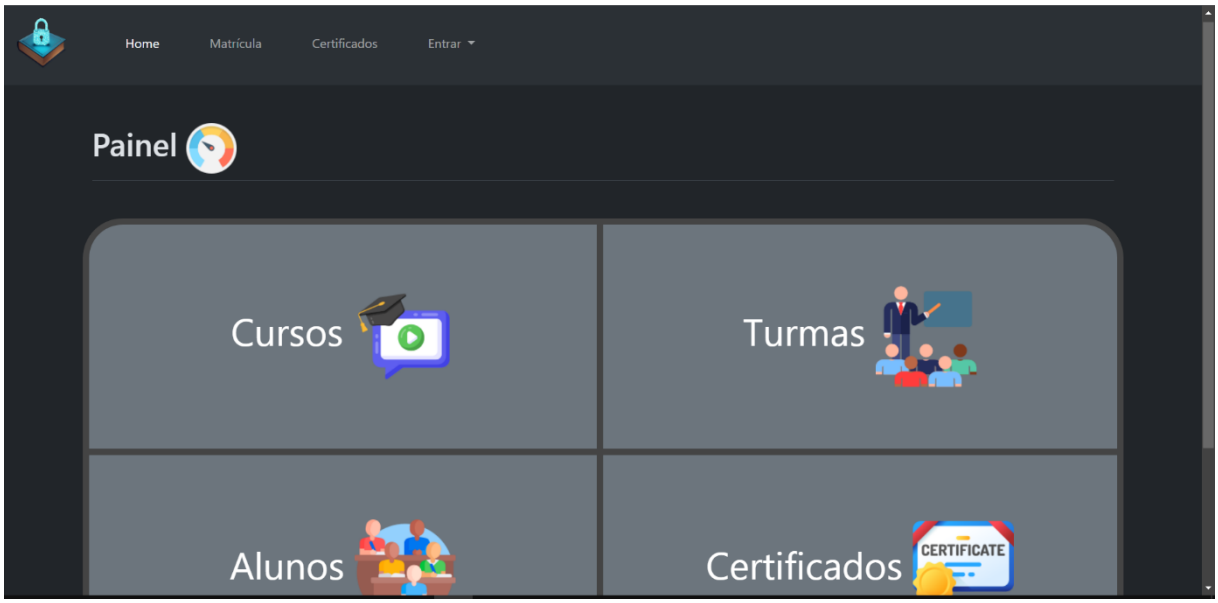
O *front-end* da aplicação tentou simular uma interface típica de uma instituição de ensino. A figura 12 mostra a tela principal da aplicação web.

Figura 12 - Página Inicial da Aplicação

Fonte: Elaborado pelo Próprio Autor

A figura 13 mostra o painel administrativo da instituição, é preciso fazer login com credenciais institucionais para poder entrar nesse painel. Cada item do painel leva a uma página que possibilita o controle de leitura, alteração, criação e remoção do respectivo item.

Figura 13 - Painel Administrativo



Fonte: Elaborado pelo Próprio Autor

4.2 Implantação do Contrato Inteligente no *Back-end*

Para integrar o *back-end* da aplicação ao *ethereum*, primeiro foi preciso criar um arquivo ".env" para armazenar alguns valores que foram usados ao longo de todo o programa. Foi preciso a chave privada da conta *Ethereum*, além da chave de API disponibilizado pela Alchemy e a chave de API disponibilizado pelo etherscan, como mostrado na figura 14.

Figura 14 - Arquivo .env

```
DATABASE_URL="postgresql://postgres:123@localhost:5434/nest?schema=public"
JWT_SECRET="super-secret"
PRIVATE_KEY="[redacted]"
PUBLIC_KEY="0x0D0820914fd462A469a72e20d312ABC669397f69"
SEPOLIA_API_KEY="ASwDCoIphWvi8zKMvihU2isjbGls6deW"
API_URL="https://eth-sepolia.g.alchemy.com/v2/ASwDCoIphWvi8zKMvihU2isjbGls6deW"
ETHERSCAN_API_KEY="KNSX22GR5R5FN99SERIAX2T679XW4ZX17F"
```

Fonte: Elaborado pelo Próprio Autor

Outro arquivo necessário é o "hardhat.config.js". É ele que compila e implanta o contrato na *Ethereum*. A configuração desse arquivo pode ser vista na figura 15.

Para implantar o contrato na rede principal da *Ethereum*, mude o nome “sepolia” para “mainnet”.

Figura 15 - Arquivo de configuração hardhat.config.js

```
/** @type import('hardhat/config').HardhatUserConfig */

require('dotenv').config();
require('@nomiclabs/hardhat-ethers');
require("@nomiclabs/hardhat-etherscan");

const { API_URL, PRIVATE_KEY } = process.env;
const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY;

module.exports = {
  solidity: "0.8.19",
  defaultNetwork: "sepolia",
  networks: {
    hardhat: {},
    sepolia: {
      url: API_URL,
      accounts: [`0x${PRIVATE_KEY}`]
    }
  },
  etherscan: {
    apiKey: ETHERSCAN_API_KEY
  }
};
```

Fonte: Elaborado pelo Próprio Autor

Para implantar o contrato inteligente, deve-se criar uma pasta chamada “contracts” na raiz do projeto e colocar o contrato inteligente dentro dela. Depois deve-se executar no terminal o comando “npx hardhat compile”. Isso gera o *bytecode* do contrato.

Para implantar o contrato é preciso rodar um código javascript que muda de acordo com os dados do certificado específico. Como a aplicação deve implantar os certificados em tempo de execução, o arquivo javascript com o código para implantar o contrato é criado em tempo de execução pelo servidor. Como mostra a figura 16, código é colocado dentro de uma variável, depois, um arquivo chamado “deploy.js” será criado com o conteúdo dessa variável e colocado dentro de uma pasta chamada scripts que está na raiz do projeto.

Para executar o código criado dinamicamente em tempo de execução foi preciso usar a função exec do nodeJS e executar comando “npx hardhat run scripts/deploy.js --network sepolia”. Se for a primeira vez que o *bytecode* estiver sendo executado na *blockchain*, será preciso fazer a publicação do contrato no

etherscan, caso contrário, só a implantação é necessária. Depois que o comando executar, o contrato inteligente estará na *blockchain*.

Figura 16 - Código para Implantar o Contrato Inteligente

```
const codigo = `
  async function main() {
    const certificadoFactory = await ethers.getContractFactory("Certificado");
    const certificado = await certificadoFactory.deploy(
      "${aluno.nome}",
      "${cpfFormatado}",
      "${curso.nome}",
      "${curso.descricao}",
      "${aluno.endereco_eth}"
    );
    console.log(certificado.address);
  }
  main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });`;
```

Fonte: Elaborado pelo Próprio Autor

4.3 Da Matrícula Até a Validação do Certificado.

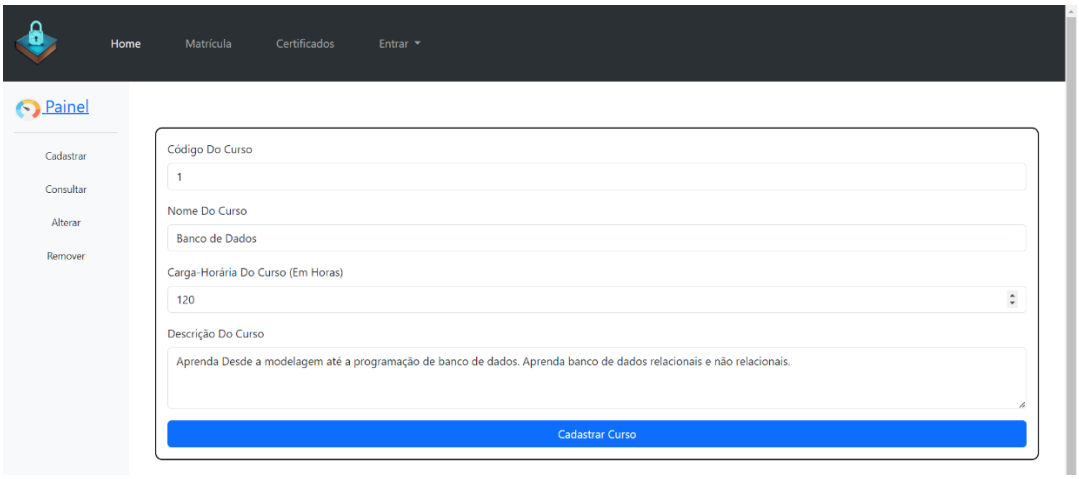
A aplicação concluiu seu objetivo com êxito, simulando um ambiente de uma instituição de ensino. Considerando um estado inicial com o banco de dados apenas com o registro do login institucional, foi possível fazer uma simulação que cobriu desde a matrícula do aluno até a validação do seu certificado na *blockchain*.

Inicialmente, com o banco de dados tendo apenas 1 registro (o login institucional). Fez-se o login com a conta institucional e foi cadastrado um curso com os dados presentes na figura 17.

Na figura 18, foi cadastrado uma turma que tem como curso oferecido o curso que foi registrado na figura 17.

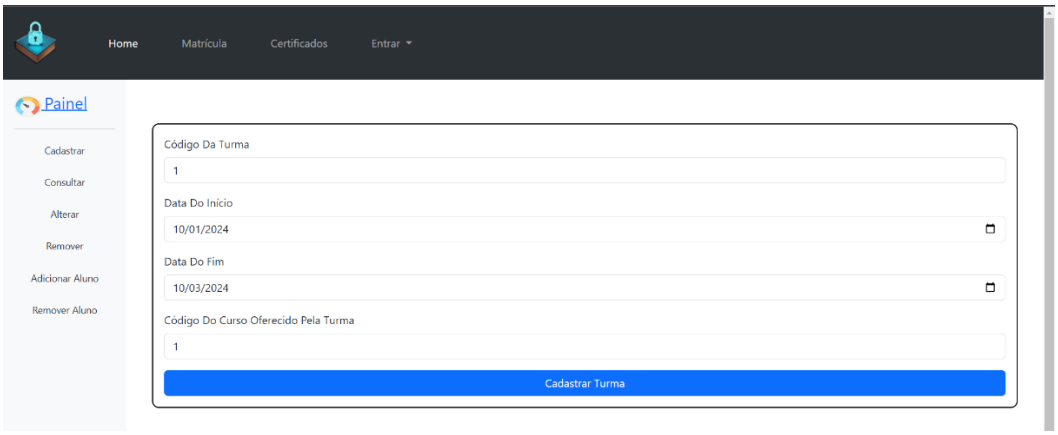
Na figura 19, a matricula de um aluno chamado jose, com o número de cpf igual a 12345678910 foi feita com sucesso. O endereço *Ethereum* desse aluno foi gerado clicando no botão “Gerar endereço *Ethereum*”. Era possível deixar esse campo em branco ou até mesmo colocar um endereço *Ethereum* que o aluno já possuía anteriormente.

Figura 17 - Cadastro de um curso



Fonte: Elaborado pelo Próprio Autor

Figura 18 - Cadastro de uma turma



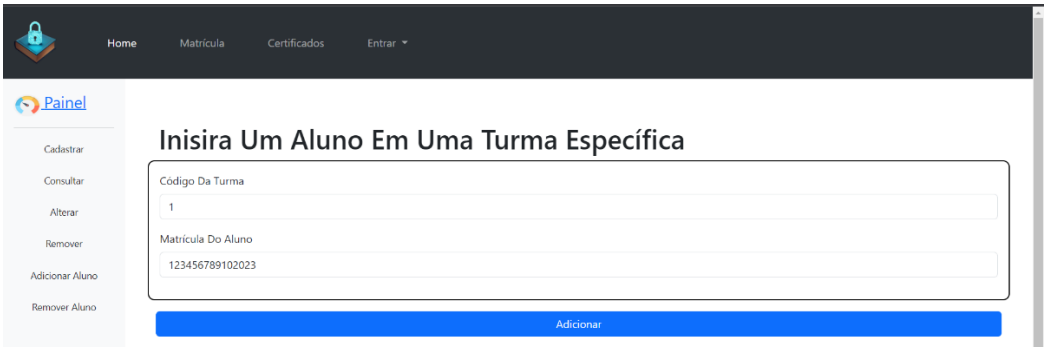
Fonte: Elaborado pelo Próprio Autor

Figura 19 - Matrícula de um aluno

Fonte: Elaborado pelo Próprio Autor

Na figura 20, é possível ver a instituição de ensino cadastrando o aluno na turma que foi criada na figura 18.

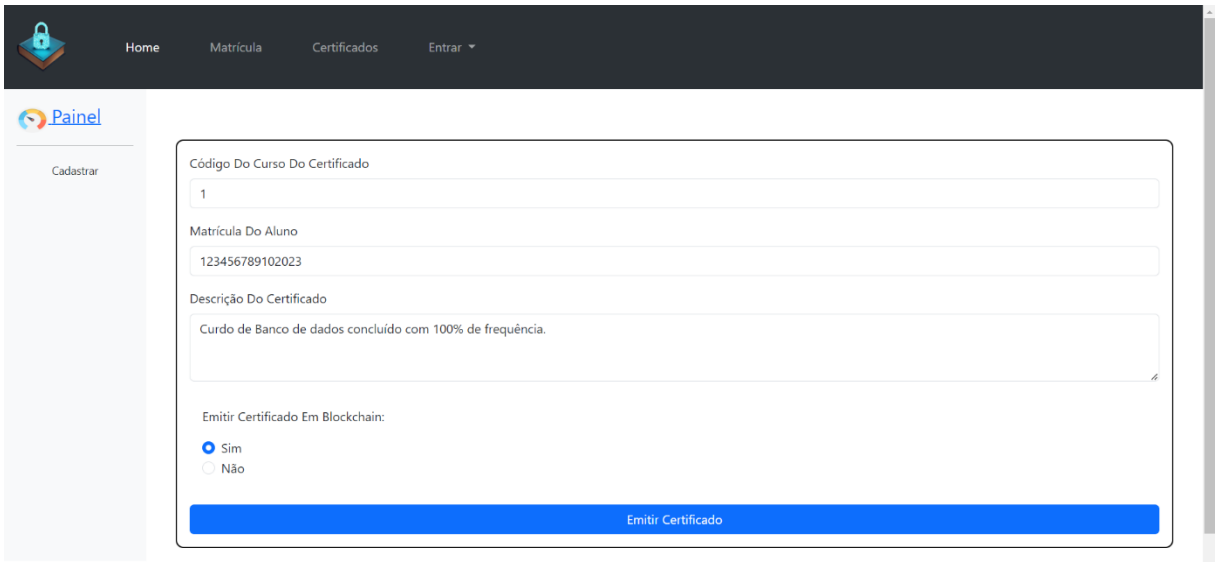
Figura 20 - Inserindo o Aluno na Turma Criada



Fonte: Elaborado pelo Próprio Autor

Na figura 21, é possível ver a instituição de ensino emitindo um certificado ao aluno cadastrado na figura 19. É possível ver que a instituição pode optar por não emitir o certificado na *blockchain*, caso em que o certificado só é armazenado no banco de dados da instituição. Caso se escolha emitir o certificado na *blockchain*, o tempo para o servidor retornar um reposta dizendo que o certificado foi emitido foi em média de 10 segundos.

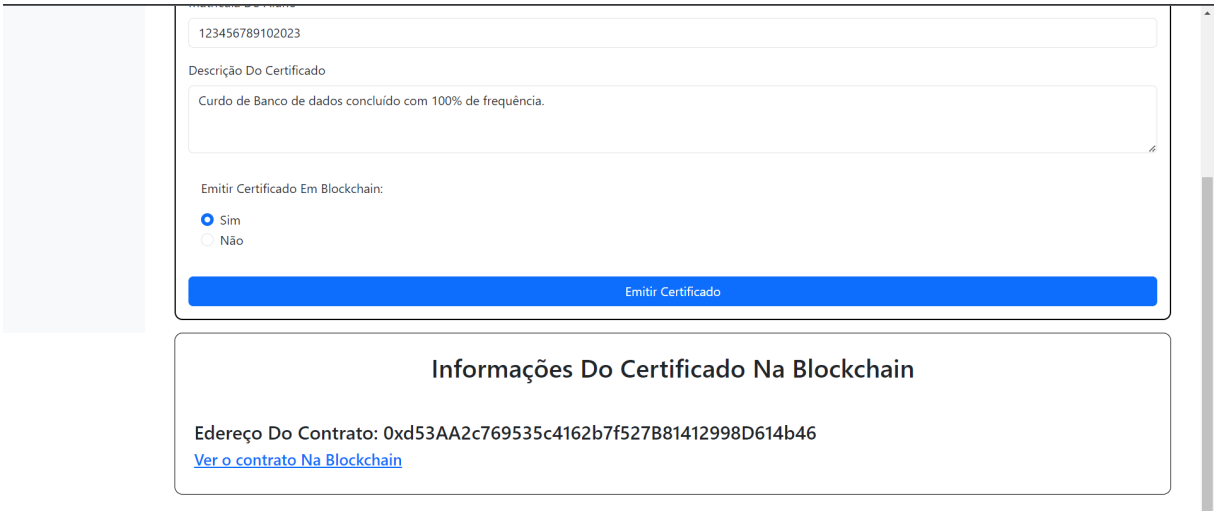
Figura 21 - Emitindo o Certificado ao Aluno



Fonte: Elaborado pelo Próprio Autor

Já na figura 22, é possível ver que o certificado foi emitido na *blockchain* com sucesso. Nesse caso, o servidor retornou o endereço do contrato e link em que se pode ver os dados do certificado. Depois de implantado, o tempo médio para que os dados do contrato estivessem na *blockchain*, foi de 40 segundos.

Figura 22 - Certificado Emitido com Sucesso



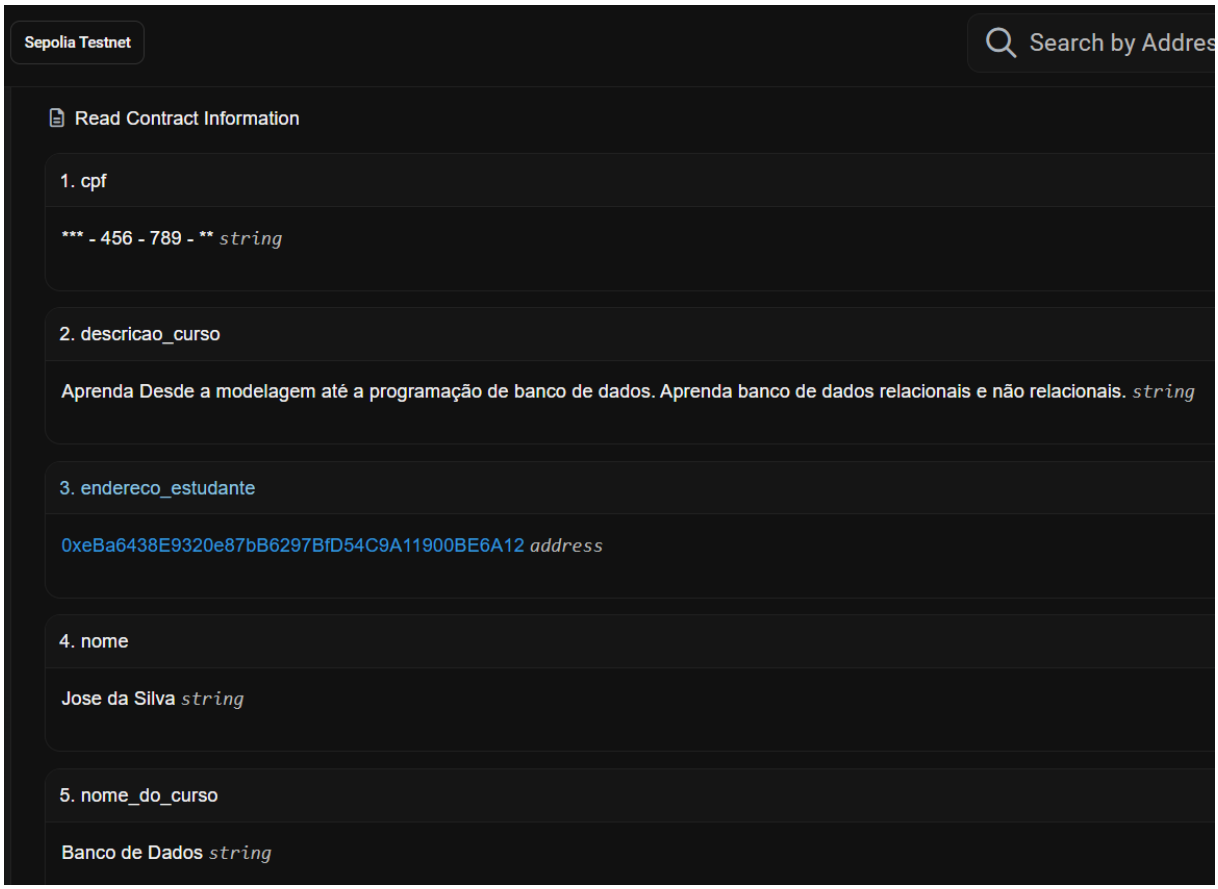
Fonte: Elaborado pelo Próprio Autor

Ao clicar no link para ver o cerificado na *blockchain*, uma página do explorador de blocos etherscan é aberta no navegador, ela mostra os dados que estão no contrato inteligente e que são, na verdade, os dados do aluno e do curso, tal como mostra a figura 23.

O certificado também foi gerado no banco de dados da instituição, estando disponível para o aluno baixá-lo quando quiser, como mostra a figura 24.

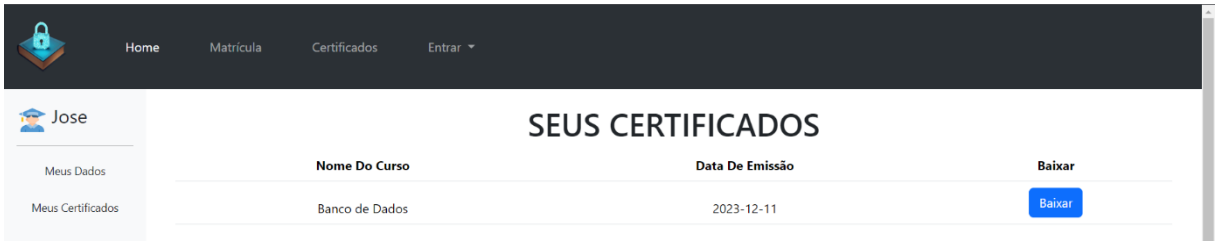
O certificado em pdf gerado pela instituição pode ser obtido ao clicar no botão “baixar”, que está na figura 24. O certificado em pdf do aluno está na figura 25. Pode-se perceber que o certificado tem o endereço do contrato inteligente na *blockchain* que o representa escrito no certificado. É por meio do desse endereço que é possível validar se o contrato é válido ou não e também acessar os dados do contrato na *blockchain*. Nesse caso específico foi colocado uma URL que já direciona para uma página do explorado de blocos que contém os dados do certificado.

Figura 23 - Informações do Certificado na *Blockchain*



Fonte: Elaborado pelo Próprio Autor

Figura 24 - Certificado Disponível para *Download*



Fonte: Elaborado pelo Próprio Autor

Figura 25 - Certificado em PDF

Certificamos que o estudante Jose da Silva concluiu o curso de Banco de Dados com uma carga-horária de 120 horas

Veja o certificado na blockchain:https://sepolia.etherscan.io/address/0xd53AA2c769535c4162b7f527B81412998D614b46#readContract

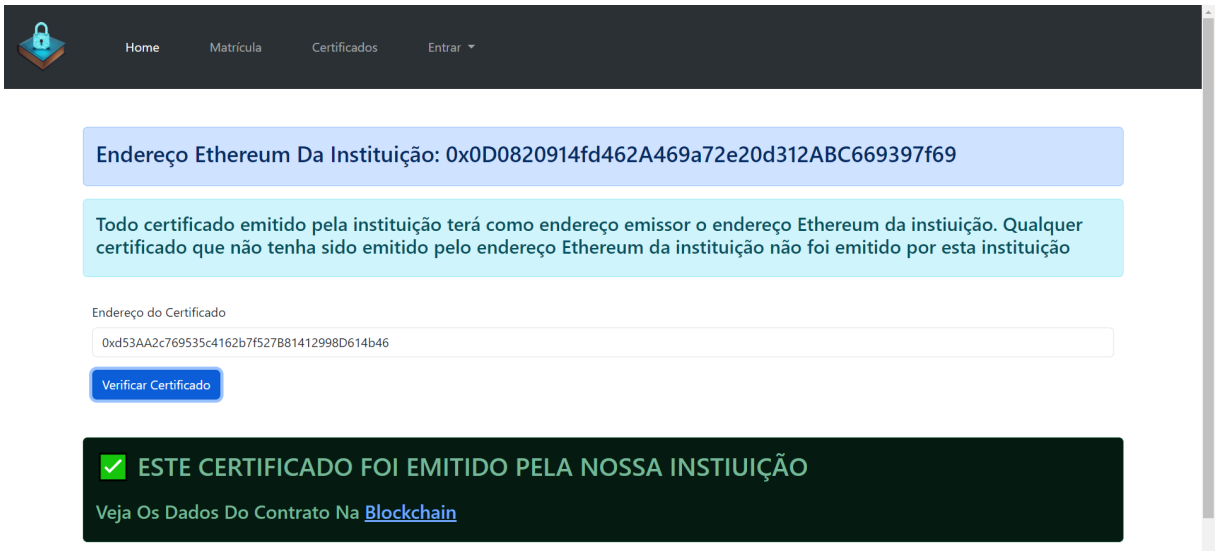
Fonte: Elaborado pelo Próprio Autor

A figura 26 mostra um certificado sendo validado no site da instituição. Vale ressaltar que não é preciso usar a infraestrutura da instituição para validar o certificado, podendo validá-lo diretamente na *blockchain*, porém a instituição de ensino pode oferecer esse serviço como forma de melhorar a experiência do usuário, que foi o caso em questão. Para validar o certificado, basta colocar o endereço *Ethereum* dele e apertar o botão “verificar certificado”.

Na figura 26, foi colocado o endereço do contrato que foi gerado na figura 22 e que é “0xd53AA2c769535c4162b7f527B81412998D614b46”. O processo de validão demorou em média cerca de 15 segundos, já que toda a *blockchain* é percorrida. Caso o endereço do criador do contrato inteligente seja o mesmo que o endereço da instituição, o certificado é válido, caso contrário, inválido.

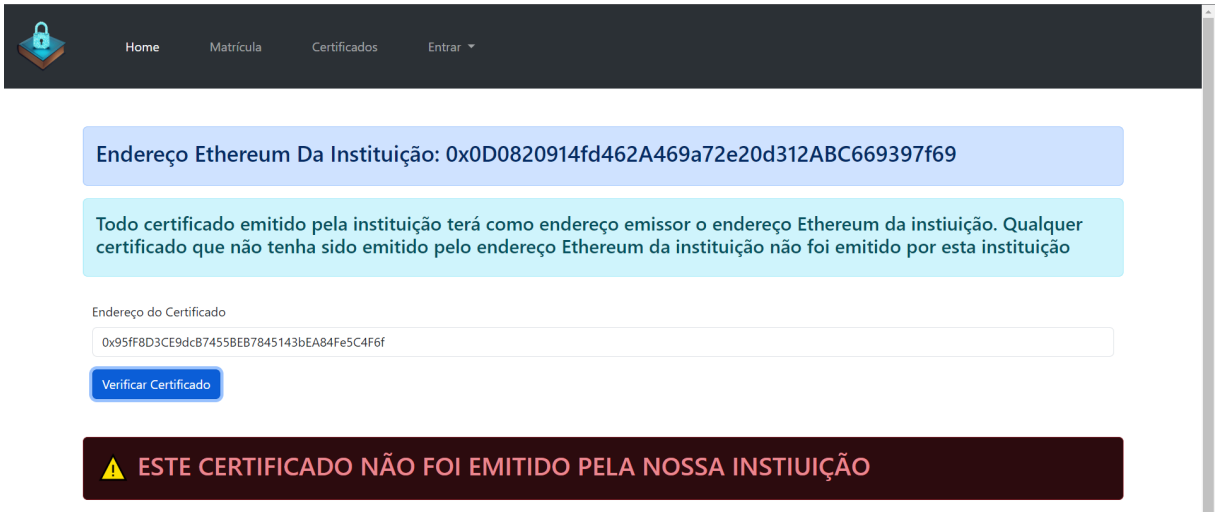
Como mostra a figura 27. Ao se colocar um outro endereço de contrato que não foi emitido pela instituição, a aplicação acusará que o certificado é inválido, já que não foi emitido pela instituição.

Figura 26 - Validando o Certificado Emitido ao Aluno Jose



Fonte: Elaborado pelo Próprio Autor

Figura 27 - Tentando Validar um Certificado Inválido



Fonte: Elaborado pelo Próprio Autor

5 CONCLUSÕES

Este trabalho propõe uma forma de se autenticar um certificado ou diploma emitido por uma instituição de ensino, de forma que sua validade esteja desvinculada da instituição que o emitiu.

De acordo com os resultados obtidos, pôde-se perceber que a *blockchain* serviu como uma espécie de banco de dados adicional, o que adicionou uma camada de redundância aos certificados. Outro ponto é que *blockchain* conseguiu fornecer autenticidade e integridade aos certificados emitidos, sendo toda a parte de validação dos mesmos feito apenas na *blockchain*, não necessitando da infraestrutura e do banco de dados da instituição nesse processo. Como no caso em questão, mesmo que não se dependa da infraestrutura da instituição, pode-se usar o site da instituição como um intermediário para acessar a *blockchain* e validar o certificado com o fim de melhorar a experiência do usuário.

Pôde-se ver que duas propriedades foram fundamentais para tornar tudo isso possível: a imutabilidade dos contratos inteligentes e as contas *Ethereum*. A imutabilidade dos contratos inteligentes garantiu que os dados registrados na *blockchain* não mudassem, o que garantiu a integridade dos certificados. Já as contas *Ethereum* possuem um endereço vinculado a cada uma delas. Ao se emitir um contrato inteligente na *blockchain*, pode-se ver qual endereço o emitiu. Dessa forma, para validar o certificado, é preciso apenas do endereço da instituição e o endereço de quem emitiu o certificado (o contrato inteligente), se o endereço da instituição for o mesmo endereço de quem emitiu o contrato inteligente, então pode-se garantir matematicamente que o certificado de fato foi emitido pela instituição.

5.1 Desvantagens e Desafios encontrados

Dentre os desafios e desvantagens desse modelo de certificação, pode-se listar:

- **Preço:** contratos inteligentes custam dinheiro para ser implantados, deve-se escolher uma *blockchain* com preço mais acessíveis. Pode-se também deixar o estudante decidir se quer ou não emitir o certificado em *blockchain*, sendo ele a arcar diretamente com os custos.

- **Imutabilidade:** apesar de trazer benefícios, também carrega desvantagens. Ao emitir um certificado errado, ainda que se possa modifica-lo no banco de dados da empresa, não será possível removê-lo da *blockchain*, sendo possível no máximo emitir um novo certificado na *blockchain* com os dados certos, mesmo assim, o certificado errado continuará existindo e será válido se aplicado o método de verificação mostrado neste trabalho.
- **Comprometimento de chave privada:** o que torna tudo isso possível são as contas *Ethereum* que são basicamente uma chave privada e um endereço. Para emitir os certificados é preciso a chave privada da instituição. Se a chave privada da instituição for comprometida, o atacante poderia emitir vários certificados em nome da instituição na *blockchain*. Uma maneira de tentar impedir esse tipo de situação seria adotar um modelo de *multisign*.
- **Interface do usuário e Experiência do usuário*:*** a falta de conhecimento sobre *blockchain* por parte dos usuários pode impedir que o usuário consiga validar o certificado diretamente na *blockchain*. Uma maneira de contornar isso seria construir interfaces que se conectam à *blockchain* para o usuário validar os certificados, está abordagem foi usada neste trabalho.
- **Conhecimento do endereço da instituição de ensino:** para validar a autenticidade do certificado, é preciso o endereço da instituição, porém se a instituição não existir mais de modo que não se consiga encontrar mais o endereço da instituição, não será possível verificar a autenticidade do certificado. Uma possível solução é a divulgação ampla do endereço da instituição enquanto ela existe.

5.2 Contribuições

Este trabalho traz vários aspectos técnicos e práticos sobre a tecnologia *blockchain* e pode ser usado como base para outros estudos acadêmicos relacionados à *blockchain*.

Este trabalho pode ser usado como fonte de inspiração para aqueles que pretendem emitir certificados ou documentos em *blockchain*.

5.3 Trabalhos Futuros

Dadas as conclusões deste trabalho, ainda é possível aumentar seu escopo. A emissão de certificados e diplomas é apenas uma espécie de um gênero maior que é a emissão de documentos em *blockchain*.

Além disso, existem outras formas de se validar certificados usando blockchain, como é caso do uso do *soulbound token* (token vinculado à alma) para emissão de certificados, caso em que se usa um *Non Fungible Token* (NFT) para representar um certificado.

Ainda é possível usar *blockchains* privadas para emitir os certificados, o que pode trazer algumas vantagens para o modelo deste trabalho, nesse caso, uma *blockchain* feita por um grupo de universidades, por exemplo, se encaixaria nesse item.

6 REFERÊNCIAS

Bitcoin Wiki. **Difficulty**. 2021. Disponível em: <https://en.bitcoin.it/wiki/Difficulty>. Acesso em: 03 mar. 2023.

Bitcoin Wiki. **Block hashing algorithm**. 2021. Disponível em: < https://en.bitcoin.it/wiki/Block_hashing_algorithm>. Acesso em: 03 mar. 2023.

Bitcoin Wiki. **Protocol documentation**. 2021. Disponível em: < https://en.bitcoin.it/wiki/Protocol_documentation>. Acesso em: 03 mar. 2023.

DURANT, Elizabeth; TRACHY, Alison. Digital Diploma debuts at MIT. **Massachusetts Institute of Technology**. 2017. Disponível em: < https://news.mit.edu/2017/mit-debuts-secure-digital-diploma-using-bitcoin-blockchain-technology-1017>. Acesso em: 30 out. 2023.

FreeCodeCamp. **How to create an Ethereum wallet address from a private key**. 2018. Disponível em: <https://www.freecodecamp.org/news/how-to-create-an-ethereum-wallet-address-from-a-private-key-ae72b0eee27b/>. Acesso em 05 nov. 2023.

KOK, Arjuna Sky. **Hands-On Blockchain for Python Developers**. BIRMINGHAM: Packt Publishing, 2019. ISBN 978-1-78862-785-6.

MIT Technology Review. Blockchain além das criptomoedas: o futuro financeiro com o DREX, o Real Digital. 2023. Disponível em: < https://mittechreview.com.br/blockchain-alem-das-criptomoedas-o-futuro-financeiro-com-o-drex-o-real-digital/>. Acesso em: 01 nov. 2023.

MORENO, Ana Carolina. et al. Tecnologia do sistema de vacinação contra a Covid impede que doses sejam apagadas sem deixar rastro. **g1**. 2023. Disponível em: < https://g1.globo.com/saude/noticia/2023/05/04/tecnologia-do-sistema-de-vacinacao-contra-a-covid-impede-que-doses-sejam-apagadas-sem-deixar-rastro.ghtml>. Acesso em: 05 jul. 2023.

NAKAMOTO, Satoshi (2008). **Bitcoin: A Peer-to-Peer Electronic Cash System**. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 03 fev. 2023.

National Institute of Standards and Technology (NIST), 2018. **Blockchain Technology Overview.** Disponível em: <https://doi.org/10.6028/NIST.IR.8202>. Acesso em: 07 de fev. 2023.

National Institute of Standards and Technology (NIST), 2020. **A Taxonomic Approach to Understanding Emerging Blockchain Identity Management Systems.** Disponível em: <https://doi.org/10.6028/NIST.CSWP.01142020>. Acesso em: 27 de ago. 2023.

National Institute of Standards and Technology (NIST), 2021. **Blockchain Networks: Token Design and Management Overview**. Disponível em: <https://doi.org/10.6028/NIST.IR.8301>. Acesso em: 10 de jun. 2023.

RAWAL, Bharat S. et al. **Implementing and Leveraging Blockchain Programming.** Singapura: Springer, 2022. ISBN 978-981-16-3412-3.

RODRIGUES, Josina dos Santos. **Blockchain: Um Novo Modelo Social e Financeiro**. Porto, 2019. 313 p. Tese (Doutorado em Sistemas, Tecnologias e Gestão da Informação) - Universidade Fernando Pessoa. Disponível em:

<https://bdigital.ufp.pt/bitstream/10284/8354/1/TD_Josina%20Rodrigues.pdf>. Acesso em: 17 mar. 2023.

SHARMEELA, C. et al. **IoT, Machine Learning and Blockchain Technologies for Renewable Energy and Modern Hybrid Power Systems**. Gistrup: River Publishers, 2023. ISBN 978-10-0082-440-7.

PRUSTY, Narayan. **Blockchain for Enterprise**. Birmingham: Packt Publishing, 2018. ISBN 978-1-78847-974-5.

KUANAR, Sanjay K. *et al*. **The Role of Iot and Blockchain**: Techniques and Applications. Palm Bay: Apple Academic Press, 2022. ISBN 978-1-00304-836-7.

GRINCALAITIS, Merunas. **Mastering Ethereum**: Implement advanced blockchain applications using Ethereum-supported tools, services, and protocols. Birmingham: Packt Publishing, 2019. ISBN 978-1-78953-137-4.

XIAO, Perry. **Practical Java Programming for IoT, AI, and Blockchain**. Indianápolis: John Wiley & Son, 2019. ISBN 978-1-119-56003-6.

UPADRISTA, Venkatesh. **IoT Standards with Blockchain**: Enterprise Methodology for Internet of Things. Slough: Apress, 2021. ISBN 978-1-4842-7271-8.

Ethereum. **The Merge.** 2022. Disponível em: <https://ethereum.org/en/roadmap/merge/>. Acesso em: 03 mar. 2023.

Ethereum. **Networks**. 2023. Disponível em: <https://ethereum.org/pt-br/developers/docs/networks/>. 10 nov. 2023.

APÊNDICES

Apêndice A – Gerador de contas *Ethereum* em NodeJS

Comandos para configurar o ambiente e baixar as bibliotecas

> npm init -y

> npm i ethereum-cryptography

Código do programa

```
const { secp256k1 } = require('ethereum-cryptography/secp256k1.js');
const { getRandomBytesSync } = require('ethereum-cryptography/random.js');
const { toHex } = require('ethereum-cryptography/utils.js');
const { keccak256 } = require('ethereum-cryptography/keccak.js');


function gerarContaEthereum() {
    const chavePrivada = getRandomBytesSync(32);
    const chavePublica = secp256k1.getPublicKey(chavePrivada, false);
    const hashChavePublica = keccak256(chavePublica.slice(1));
    const endereco = hashChavePublica.slice(-20);

    console.log('Chave Privada: 0x' + toHex(chavePrivada));
    console.log('Endereço: 0x' + toHex(endereco));
}


function main() {
    gerarContaEthereum();
}

main();
```

Use o comando “node <nome_do_seu_arquivo>” para rodar o programa

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 ● Setor Universitário
Caixa Postal 86 ● CEP 74605-010
Goiânia ● Goiás ● Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br ● reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante VALTECI MARCELINO COELHO JUNIOR do Curso de Ciência de Computação, matrícula 2019.1.0028.00330, telefone: (62) 98525-7161 e-mail valtecijunior@gmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado **USANDO BLOCKCHAIN PARA EMITIR CERTIFICADOS E DIPLOMAS**, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 14 de dezembro de 2023.

Assinatura do autor: _____

Nome completo do autor: VALTECI MARCELINO COELHO JUNIOR

Assinatura do professor–orientador: _____

Nome completo do professor-orientador: Daniel Corrêa da Silva