Curso de Sistemas de Informação

Universidade Estadual de Mato Grosso do Sul

Sistema de registro de certificados acadêmicos em blockchain

Guilherme Paiva Nogueira

Dr. Cleber Valgas Gomes Mira (Orientador)

Dourados

2025

Sistema de registro de certificados acadêmicos em blockchain

Guilherme Paiva Nogueira

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Guilherme Paiva Nogueira e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 07 de novembro de 2025.

Prof. Dr. Cleber Valgas Gomes Mira (Orientador)

N712s Nogueira, Guilherme Paiva.

Sistemas de registro de certificados acadêmicos em blockchain / Guilherme Paiva Nogueira. – Dourados, MS: UEMS, 2025.
90 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2025.
Orientador: Prof. Dr. Cleber Valgas Gomes Mira.

1. Certificados acadêmicos digitais 2. *Blockchain* 3. Sistemas descentralizados 4. Segurança I. Mira, Cleber Valgas Gomes II. Título

CDD 23. ed. - 005.1

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Bruna Peruffo Vieira – CRB 1/2959

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Sistema de registro de certificados acadêmicos em blockchain

Guilherme Paiva Nogueira
07 de novembro de 2025

Banca Examinadora:

Prof. Dr. Cleber Valgas Gomes Mira (Orientador)
Área de Computação – UEMS

Prof. Dr. Fabrício Sérgio de Paula
Área de Computação – UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação – UEMS

AGRADECIMENTOS

Chegando a este momento tão significativo, não posso deixar de expressar minha profunda gratidão a todos que foram essenciais nesta jornada.

À minha família, meus primeiros e mais sinceros agradecimentos. Vocês são meu alicerce, meu porto seguro. Obrigado pelo amor incondicional, pela paciência infinita e por todos os sacrifícios, grandes e pequenos, que fizeram para que eu chegasse até aqui. Se hoje colho os frutos deste esforço, é porque vocês regaram a semente com seu apoio inabalável. Esta conquista é tão minha quanto de vocês.

Ao meu orientador, Prof. Dr. Cleber Valgas Gomes Mira, meu sincero agradecimento pela liderança inspiradora e pela sabedoria compartilhada ao longo deste percurso. Sua visão crítica e seu apoio constante foram indispensáveis para a concretização deste trabalho. Foi um privilégio contar com sua orientação.

Aos meus amigos, minha rede de apoio e alegria. Obrigado por cada palavra de incentivo, por cada risada que aliviou a tensão e por toda a compreensão nos momentos em que eu estava mais ausente. Vocês tornaram a caminhada mais leve e feliz. Obrigado por celebrarem cada vitória comigo e por me apoiarem nos tropeços. Saber que podia contar com vocês fez toda a diferença.

A cada um de vocês, o meu eterno e mais sincero obrigado.

"Sair da caverna é difícil, pois a luz da verdade cega os olhos acostumados à escuridão."

(Platão)

RESUMO

O avanço da tecnologia trouxe diversos benefícios para a sociedade e cada vez mais informações pessoais estão sendo armazenadas digitalmente em computadores ou celulares. Com base nisso, certificados digitais acadêmicos também passaram por essa transição da sua forma física para o modelo digital. Atualmente os certificados digitais acadêmicos já são uma realidade, desde sua implementação, eles seguem normas pré estabelecidas e necessitam que uma terceira entidade faça as validações de modo a autenticar o documento. O trabalho foi desenvolvido em torno desse problema, desenvolver um sistema de registro de certificados acadêmicos digitais sem a necessidade de uma terceira entidade de confiança para verificar sua autenticidade. Para desenvolver esse trabalho, foi necessário utilizar uma rede distribuída chamada *blockchain*. Essa tecnologia garante que não é necessário ter um terceiro agente envolvido nas validações, junto à isso, assegurando imutabilidade, transparência e total segurança.

Palavras-chave: Certificados acadêmicos digitais, *Blockchain, Ethereum*, Sistemas descentralizados, Segurança.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

MEC - Ministério da Educação

IES - Instituição de Ensino Superior

RSA - Rivest-Shamir-Adleman

XML - Extensible Markup Language

ICP - Infraestrutura de Chaves Públicas

AC - Autoridade Certificadora

PoW - Proof Of Work

PoS - Proof Of Stake

DPoS - Delegated Proof Of Stake

P2P - Peer-to-Peer

BFT - Byzantine Fault Tolerance

UNIC - Universidade de Nicosa

MIT - Massachusetts Institute of Technology

EVM - Ethereum Virtual machines

UFPB - Universidade Federal da Paraíba

UFSC - Universidade Federal de Santa Catarina

ETH - Ether

GWEI - Giga-Wei

CBC — Cipher Block Chaining

CTR — Counter (Mode)

GCM — Galois/Counter Mode

SUMÁRIO

Capítulo 1

Introdução

A ampla adoção da tecnologia resultou na resolução bem-sucedida de diversos desafios socias através da implementação de soluções tecnológicas inovadoras. Um exemplo é a gestão de certificações acadêmicas por meio de documentos digitais.

A implementação de documentos digitais foi um meio de se preservar contra danos externos do ambiente, assegurar contra possíveis fraudes que ocorrem frequentemente nesse meio e também ter uma fácil busca e organização dos documentos. Assim como ocorre para certos tipos de documentos feitos de maneira tradicional, alguns documentos digitais podem precisar de um intermediador que faça uma validação que garanta a sua confiabilidade. A tecnologia que é aplicada nesse caso é a assinatura digital, implementada com criptografia de chave pública e chave privada, que são geradas por uma autoridade central (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2016).

No âmbito da educação, as certificações acadêmicas são consideradas como documentos de grande relevância para quem as possuem. Segundo o Ministério da Educação (MEC), todos os dados e registros acadêmicos devem ser mantidos pela Instituição de Ensino Superior (IES) (BRASIL, 1991). Diante disso, todos esses dados estão sujeitos a uma eventual perda parcial ou total desses registros. Um episódio conhecido no Brasil foi o caso "Gama Filho", a instituição foi descredenciada por ordem do MEC e teve que fechar as portas da universidade, assim prejudicando de diversas formas os estudantes como a perda de seu histórico acadêmico, impedindo a comprovação de graduação ou a transferência para outras universidades (SANTOS, 2023).

Outra situação problemática relacionada a emissão de certificações acadêmicas são as fraudes. Os acontecimentos envolvendo a segurança dos certificados são frequentes.

Existem quadrilhas que são especialistas nesse ramo de fraude, realizam vendas para diversos tipos de pessoas e áreas especificas como diplomas de professores e alunos de um determinado curso (Polícia Federal, 2025).

Diante desse cenário, este trabalho adota a tecnologia *blockchain* para registrar e verificar a emissão, a verificação e a revogação da certificação acadêmica. A *blockchain* mantém um registro compartilhado entre várias partes, com marcação de tempo e resistente a alterações, o que favorece a integridade, a autoria e a auditoria por terceiros (NARAYANAN *et al.*, 2016). Em um banco de dados tradicional, a confiança fica concentrada na equipe que administra o sistema local, o que cria um ponto único de falha e de controle (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2016). Com *blockchain*, o resumo criptográfico do documento e os metadados essenciais ficam ancorados em um livro digital replicado, tornando mudanças posteriores detectáveis e a validação externa mais simples. O modelo pode operar em rede autorizada entre instituições ou em rede pública, com dados pessoais guardados fora da cadeia e apenas identificadores e provas criptográficas expostos para atender a requisitos de privacidade (NAKAMOTO, 2008).

O sistema desenvolvido neste trabalho pode ser uma possibilidade para instituições acadêmicas utilizarem como forma de gerar e manter seguros todos os certificados de alunos e professores, para previnir fraudes, manter a privacidade dos dados e cumprir as exigências legais e regulatórias das certificações acadêmicas digitais.

1.1 Justificativa

Os certificados acadêmicos digitais com o passar dos anos detém um valor de grande relevância para os alunos e profissionais de qualquer área, seja a conclusão de um curso superior ou até mesmo a participação em um evento sobre determinado assunto específico. Dessa maneira, a segurança e legitimidade desse certificado é de severa importância para todos os agentes envolvidos na certificação acadêmica, seja para o propietário do certificado, o emissor do certificado ou de uma pessoa que precisa verificar se o certificado é verdadeiro e não foi modificado.

Assegurar esses aspectos é essencial para preservar a confiança no processo formativo, uma vez que certificados inválidos ou adulterados podem comprometer trajetórias profissionais, prejudicar instituições e gerar insegurança para recrutadores e órgãos

avaliadores. Ao garantir autenticidade e integridade, a certificação digital fortalece a credibilidade acadêmica, reduz vulnerabilidades e contribui para um ecossistema educacional mais transparente e confiável.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um sistema de registro e geração de certificados acadêmicos, visando a legitimidade e preservação dos documentos digitais por meio da tecnologia *Blockchain*.

Os objetivos específicos são:

- Estudar o funcionamento dos certificações acadêmicas digitais;
- Estudar a tecnologia *Blockchain* para utilização em certificações acadêmicas digitais;
- Projetar e especificar um sistema para a gestão de certificados acadêmicos digitais;
- Implementar um sistema de registro de certificados acadêmicos digitais;
- Realizar testes de emissão, validação e revogação de certificados acadêmicos digitais.

No Capítulo 2 é discutido o processo de certificados acadêmicos digitais, a tecnologia *blockchain* e sua versão *Ethereum* e também é discutido quais são os *frameworks* e bibliotecas que serão utilizados. No Capítulo 3 são apresentados os principais elementos do desenvolvimento do sistema, juntamente com imagens relevantes do projeto. No capítulo 4 é citado os testes feitos após a finalização do sistema. No capítulo 5 são relatadas as conclusões feitas a partir do estudo e implementação do sistema de geração de certificados acadêmicos digitais utilizando a tecnologia *blockchain*.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta a revisão bibliográfica que fundamenta o desenvolvimento do sistema proposto, reunindo conceitos, modelos e tecnologias pertinentes ao tema. São abordados aspectos relacionados aos processos de certificação acadêmica, aos fundamentos e à evolução da tecnologia blockchain, bem como às implicações da atuação de intermediários em ambientes digitais. Por fim, são mencionadas as principais ferramentas utilizadas para a implementação da solução desenvolvida.

A **Seção 2.1** descreve o processo de certificação acadêmica nas modalidades em papel e digital, detalhando as etapas desde a criação até a disponibilização do documento ao usuário final.

A **Seção 2.2** apresenta os fundamentos da tecnologia *blockchain*, suas principais características, estrutura de dados e funcionalidades relevantes ao problema estudado.

A **Seção 2.3** discute o papel de intermediários na certificação digital e os riscos associados à dependência de uma terceira entidade, motivando alternativas de validação verificáveis por terceiros.

A **Seção 2.4** apresenta o início da segunda geração de *blockchain* com a chegada do *Ethereum*. A seção explica qual é a grande novidade dessa tecnologia e quais são as suas vantagens.

A **Seção 2.5** apresenta as tecnologias, bibliotecas e *frameworks* selecionados para o desenvolvimento do sistema proposto.

2.1 Certificação acadêmica digital

Nos últimos anos, os documentos digitais ganharam importância no cotidiano, impulsionados pela busca de segurança e confiabilidade na sua circulação por meios eletrônicos, como computadores e telefones celulares. Aborda-se aqui a certificação acadêmica digital, entendida como o conjunto de procedimentos e registros eletrônicos que comprovam a conclusão ou a participação em atividades e eventos acadêmicos, com foco em torná-los mais seguros e práticos (FRIEDRICH; MEDINA, 2007).

No meio acadêmico, a adoção da certificação acadêmica digital traz vantagens significativas quando comparada ao modelo físico tradicional. Entre esses benefícios, destacam-se o baixo custo unitário de emissão, já que não requer papel de segurança, tintas especiais ou processos de acabamento, e a eliminação de despesas logísticas relacionadas a armazenamento, transporte, postagem e seguro. A emissão de segunda via torna-se imediata, sem necessidade de reimpressão ou nova conferência manual, e a verificação por terceiros pode ocorrer de forma instantânea, sem taxas cartoriais ou deslocamentos. Além disso, reduz-se de maneira expressiva a demanda por mão de obra administrativa dedicada a etapas manuais de assinatura e controle, ao mesmo tempo em que se eliminam riscos de perda ou extravio do documento físico. O processo digital também garante prazos mais curtos para disponibilização do certificado, melhorando a experiência do egresso e aumentando a eficiência institucional (MIRANDA, 2019).

Além das vantagens operacionais e logísticas, a certificação acadêmica digital também se destaca pela maior segurança em comparação ao modelo físico tradicional. A preocupação com fraudes permanece como um desafio significativo, especialmente diante de episódios recentes, como a operação da Polícia Federal que investigou a falsificação de certificados acadêmicos físicos em diferentes áreas profissionais (BRASIL, 2023). Esse tipo de ocorrência evidencia a fragilidade dos documentos em papel e seus riscos para instituições e egressos. A certificação digital baseada em criptografia, por sua vez, oferece mecanismos mais robustos de proteção, assegurando autenticidade, integridade e verificabilidade independente. Ao vincular a identidade da instituição emissora a um registro imutável e permitir conferência pública do conteúdo, essa abordagem reduz a probabilidade de adulterações, facilita processos de auditoria e preserva a confiabilidade do histórico acadêmico ao longo do tempo(DORNELES; CORRÊA,

2013).

Um exemplo de um padrão de *certificado digital* é o padrão X.509 (GUILHEN, 2006). Esse modelo é definido como uma estrutura de dados que vincula de modo confiável a chave pública a um sujeito por meio da assinatura de uma Autoridade Certificadora, incluindo campos como número de série, emissor, período de validade, sujeito, informação da chave pública e extensões que indicam usos e restrições; a confiança decorre da verificação criptográfica da assinatura e da cadeia de certificação (STALLINGS, 2022; COOPER *et al.*, 2008). No contexto brasileiro, essa definição é complementada por requisitos normativos que disciplinam papéis institucionais, políticas e práticas de certificação, identificação do titular, critérios de algoritmos e mídia de guarda, bem como procedimentos de revogação e auditoria, conferindo validade jurídica ao certificado quando emitido segundo as regras da ICP-Brasil (Instituto Nacional de Tecnologia da Informação (ITI), 2023).[1]

Diante do cenário de identificação baseada em certificados e chaves públicas, convém apresentar primeiro a criptografia simétrica(STALLINGS, 2022). Nessa abordagem, a mesma chave secreta, compartilhada entre as partes, é utilizada para cifrar e para decifrar as informações, o que favorece alto desempenho na proteção de grandes volumes de dados. A criptografia simétrica organiza-se em cifras de bloco e cifras de fluxo. As cifras de bloco operam dados em blocos fixos e dependem do modo de operação escolhido, como *Cipher Block Chaining* (CBC), *Counter Mode* (CTR) ou *Galois Counter Mode* (GCM). As cifras de fluxo processam os dados de forma contínua a partir de um gerador de sequência pseudoaleatória. Em ambos os casos, a confidencialidade decorre do sigilo e do gerenciamento adequado da chave, além da seleção criteriosa de parâmetros e construções (MENEZES; OORSCHOT; VANSTONE, 2018). A principal vantagem de utilizar esse sistema de criptografia é sua simplicidade e rapidez para seus usos em processos criptográficos, no entanto a sua desvantagem se dá pela sua forma de utilização: como a chave tem que ser compartilhada para o transmissor e o receptor, durante esse processo a chave pode ser interceptada para fins de fraudes, portanto é fundamental que haja um canal de comunicação confiável para o compartilhamento da mesma (OLIVEIRA, 2012; CORMEN *et al.*, 2012).

[1] O *certificado digital* identifica o titular e viabiliza a verificação de assinaturas. Não se confunde com o *documento de certificação acadêmica digital* — por exemplo, um diploma — que é o conteúdo assinado e cuja autenticidade é verificada com o auxílio do certificado digital.

Já o modelo de criptografia assimétrica foi criado pelo matemático Clifford Cocks em meados de 1970, no qual poucos anos depois foi definido como algoritmo de criptografia *Rivest Shamir Adleman* (RSA) (STALLINGS, 2022). A criptografia assimétrica utiliza um par de chaves matematicamente relacionadas, composto por uma chave pública e uma chave privada. A chave pública é divulgada livremente e permite que qualquer parte cifre uma mensagem destinada ao titular do par, enquanto a chave privada permanece sob controle exclusivo do titular e é utilizada para decifrar o conteúdo recebido. O par de chaves também possibilita assinaturas digitais: o emissor aplica sua chave privada para assinar um documento, e qualquer interessado verifica a autenticidade com a chave pública correspondente. Esse arranjo elimina a necessidade de compartilhar segredos ao estabelecer comunicações seguras, viabiliza a distribuição aberta de chaves e fundamenta infraestruturas de certificação, nas quais autoridades reconhecidas atestam a vinculação entre uma chave pública e a identidade de seu titular (MORIARTY *et al.*, 2016). Em termos práticos, a segurança depende do sigilo absoluto da chave privada, da robustez dos algoritmos escolhidos e de procedimentos adequados de geração, armazenamento e revogação de chaves. A segurança desse modelo se baseia na dificuldade de fatoração de números primos grandes, que dependendo do tamanho exigiria um trabalho que levaria muitos séculos. No Brasil, esse modelo de criptografia é utilizado pela ICP-Brasil para a emissão de certificados digitais (OLIVEIRA, 2012).

Os certificados acadêmicos digitais utilizam assinaturas digitais baseadas em criptografia assimétrica, conferindo autenticidade e integridade ao documento eletrônico, com valor jurídico equivalente ao da assinatura manuscrita quando emitidos conforme as políticas e normas vigentes de cada jurisdição e infraestrutura de certificação. As assinaturas digitais não se limitam a certificados e podem ser aplicadas a mensagens, arquivos de áudio e vídeo e transações financeiras. No processo, a assinatura vincula de forma inequívoca o certificado digital do signatário, tipicamente um certificado X.509 emitido por uma Autoridade Certificadora, ao documento assinado. A verificação é realizada com a chave pública presente nesse certificado, enquanto a assinatura é produzida com a chave privada correspondente, mantida sob controle exclusivo do titular (FIPS, 2000).

Outra tecnologia que é implementada nos certificados digitais são as funções *Hash* (STALLINGS, 2022) que mapeiam uma cadeia de dados variáveis e um valor único e de tamanho fixo. Existem diversos tipos diferentes de *Hash*, cada um com suas ca-

racterísticas e propósitos específicos. Os mais conhecidos são da família MD[2] no qual tem por característica uma cadeia de 16 *bytes*. Essas funções são muito utilizadas para acelerar o processo de assinaturas digitais em documentos, podendo ser utilizadas em outros recursos como correio eletrônico, senhas, chaves criptográficas, entre outros. Para verificar a integridade de um documento que é acompanhado pelo seu valor de *Hash*, nós podemos calcular o valor do *Hash* para este documento e verificar se ele é diferente do valor de *Hash* original, e logo, há uma grande possibilidade deste documento estar modificado ou corrompido (ADÃO; SILVEIRA; SILVEIRA, 2015).

A crescente adoção de certificações acadêmicas digitais tem sido uma importante aliada de diversas instituições de ensino no mundo todo. No Brasil a certificação digital existe desde 2001, porém só em 2006 obteve uma progressão positiva com a aprovação de uma lei que tornou legalmente válida a autenticação de documentos por cetificados digitais (KAZIENKO *et al.*, 2003). No meio acadêmico as instituições que costumam utilizar esse modelo digital dos certificados são as universidades, institutos de pesquisa, plataformas de ensino online e entre outras. Para essas instituições implementarem por exemplo os diplomas de conclusão de curso usando os certificados digitais, o Ministério da Educação exige algumas normas para serem seguidas na emissão e registro desses diplomas no formato digital. Uma dessas normas por exemplo é o artigo nº 6 da portaria nº 554 (ABMES, 2023), na qual é especificado que o diploma digital deve ser emitido no formato *Extensible Markup Language* (*XML*), o *XML* diploma digital é composto por dois arquivos interligados por um código, um arquivo com os dados privados e outro com os dados públicos (ABMES, 2023). Outro ponto importante para adoção do meio digital na expedição de diplomas no Brasil é sobre as diretrizes de certificação digital que devem atender o padrão da Infraestrutura de Chaves Públicas Brasileira, a ICP-Brasil. Essa instituição é ligada ao Estado, dessa forma utilizam um padrão de cadeia hierárquica de confiança, logo atendendo os requisitos legais e o controle rígido da cadeia do ICP-Brasil (COPALO, 2003).

Atualmente existem duas maneiras de emitir e gerar diplomas digitais utilizando os certificados digitais, a maneira centralizada e descentralizada. Os dois modelos utilizam a tecnologia de criptografia assimétrica. No modelo centralizado a instituição responsável pela emissão do diploma precisará sempre de uma autoridade certificadora (AC). Logo após a escolha de uma AC de confiança, a instituição deve solicitar a

[2]Os algoritmos de hash da família MD, como o MD5, são amplamente conhecidos, mas possuem vulnerabilidades que reduzem sua segurança. Por isso, os algoritmos SHA, especialmente o SHA-256, são os mais recomendados, pois oferecem maior robustez e menor risco de colisões.

mesma uma emissão de um modelo de certificado digital, dessa maneira a AC irá validar e depois emitir o par de chaves criptográficas para a instituição utilizar. Com o par de chaves gerado, a instituição incluirá a chave pública no certificado para validar a autencidade das assinaturas digitais e a chave privada será mantida em segredo para os documentos digitalmente. Com os dados da indivíduo no certificado digital, a AC poderá assim então emitir o certificado para a instituição, que logo armazenará em um servidor ou em algum dispositivo (PEREIRA *et al.*, 2017).

Na busca de reduzir a dependência de uma autoridade única, ganharam espaço modelos descentralizados baseados em *blockchain*. Nessa abordagem, a emissão e a verificação se apoiam em um registro replicado e resistente a alterações, o que permite validar a autenticidade mesmo que um participante venha a encerrar suas atividades: enquanto o registro e as chaves válidas permanecerem disponíveis na rede, terceiros continuam capazes de conferir a veracidade dos títulos (NAKAMOTO, 2008; NARAYANAN *et al.*, 2016).

A *blockchain* é um livro digital replicado e encadeado por funções de *hash* cuja transparência decorre da possibilidade de auditoria independente: as partes autorizadas podem verificar o histórico e o estado atual por meio das provas criptográficas registradas (NARAYANAN *et al.*, 2016). O consenso entre os participantes estabelece uma ordem compartilhada dos eventos e dificulta alterações retroativas, fortalecendo a confiabilidade do registro (CROSBY *et al.*, 2016).

Modelos centralizados para registro e validação de diplomas tendem a concentrar confiança e operação em um único repositório, criando pontos de falha, custos de coordenação e barreiras à auditoria por terceiros. Em contraste, arranjos com *blockchain* possibilitam validação independente e trilhas de auditoria compartilhadas. Para atender a requisitos de privacidade, dados pessoais não precisam ser publicados no registro: armazena-se na cadeia apenas identificadores e provas criptográficas, mantendo o conteúdo sensível em sistemas controlados pela instituição (ZYSKIND; NATHAN; PENTLAND, 2015). Essa arquitetura tem sido considerada promissora para credenciais acadêmicas, ao adicionar uma camada de verificabilidade pública ao processo (DUBROWSKY, 2019).

2.2 *Blockchain*

A tradução literal da tecnologia *Blockchain* é "cadeia de blocos". Por definição é "um livro-razão distribuído que fornece uma maneira das informações serem registradas e compartilhadas por uma comunidade" (GRECH, 2017). Sua história começa em meados dos anos 90 com o criptógrafo David Chaum, que iniciou o desenvolvimento de um sistema de pagamentos digitais de forma totalmente autônoma e sem a interferência de intermediários. Porém, não se obteve o sucesso esperado por se passar em uma época onde o comércio on-line não era muito conhecido ainda. Contudo, somente em 2008 foi publicado o conhecido *white paper* que foi intitulado como "Bitcoin: Um Sistema de Dinheiro Eletrônico *Peer-to-Peer*" por um pessoa ou um grupo com o pseudônimo de Satoshi Nakamoto (SANTOLIM, 2020; NAKAMOTO, 2008).

As propiedades fundamentais que compõem a estrutura de dados de uma *Blockchain* são (GREVE *et al.*, 2018):

- Descentralização: não existe uma entidade centralizadora controlando as aplicações e participantes da rede *blockchain*, ou seja, o controle da rede é distribuído e as transações são validadas através de um processo de consenso realizado entre os nós desse ambiente, sem a necessidade de uma entidade intermediária confiável;

- Disponibilidade e Integridade: a rede *blockchain* é composta por vários nós, em que os dados das transações são replicados em cada nó de maneira segura. Dessa forma o sistema se mantém disponível e consistente;

- Transparência e Auditabilidade: o livro de registros das transações é público e todo mundo pode verificar os dados. Além disso, os códigos fontes da tecnologia são abertos, o que permite que o sistema seja transparente e auditável;

- Imutabilidade e Irrefutabilidade: após um dado ser registrado na blockchain ele se torna imutável, ou seja, não é possível realizar modificações no mesmo. Com isso, as transações não podem ser refutadas e atualizações podem ser possíveis através de novas transações;

- Privacidade e Anonimidade: não existe terceiros envolvidos com acesso ou controle dos dados dos usuários. Através dos mecanismos de assinatura digital e

criptografia assimétrica (par de chaves pública e privada) as transações são até certo ponto anônimas, considerando o endereço dos envolvidos na *blockchain*;

- Desintermediação: a integração entre diversos sistemas de forma direta e eficiente é possível através dessa tecnologia, permitindo a eliminação de intermediários e simplificando o projeto dos sistemas e processos;
- Cooperação e Incentivos: oferece incentivos baseados na teoria dos jogos para cooperação entre os membros da rede.

O *Blockchain* recebe esse nome por conta da maneira que ele armazena os dados que são contidos na estrutura de transações em blocos que estão ligados entre si para formar uma cadeia; dessa forma cada bloco contém um registro de data e hora, um valor de *hash* do bloco anterior, um valor *hash* do bloco atual e seus dados contidos neste bloco. O valores *hash* do primeiro bloco chamado de *genesis block* são únicos e é com base nisso que garantimos a integridade dos valores contidos nele, pois, caso os valores de *hash* fossem alterados, os valores *hash* dos próximos blocos teriam que ser alterados também (CAGLIARI; SILVA; SANTOS, 2022).

A **Figura 1** representa um bloco que pode ser dividido em dois campos, o cabeçalho e o corpo. O corpo contém um contador e as transações, enquanto o cabeçalho possui o seguinte conjunto de dados (ZHENG *et al.*, 2017):

a) *block version*: contém as regras que foram utilizadas para validar o bloco;

b) *merkle root*: contém o hash de todas transações inclusas no bloco;

c) *timestamp*: data exata, no formato Unix, em que o bloco foi registrado;

d) *nBits* ou *difficulty target*: dificuldade alvo do algoritmo de prova-de-trabalho utilizada no bloco;

e) *nonce*: um contador utilizado para o cálculo do cabeçalho do bloco;

f) *previous block hash*: referência ao identificador do bloco anterior.

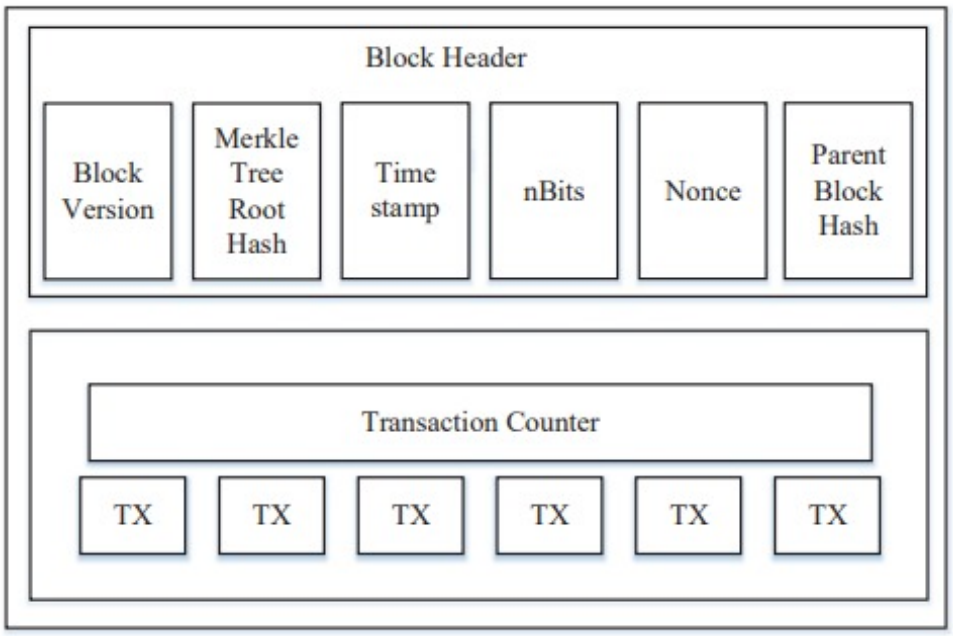


Figura 1: estrutura de um bloco da *blockchain*. Imagem adaptada do trabalho (ZHENG *et al.*, 2017)p.2.

O corpo de uma *blockchain* contém as informações sobre as transações e outros dados que estão sendo registrados naquele bloco. O elemento mais comum encontrado no corpo de uma *blockchain* são as transações e elas são representadas por ações que ocorrem dentro da *blockchain*. O contador é uma métrica que registra o número de transações ou outros dados presentes naquele bloco específico ou em toda a *blockchain*. Em certos modelos de *blockchain*, o corpo do bloco pode conter intruções para a execução de contratos inteligentes.

A **Figura 2** apresenta um registro de transações de uma cadeia de blocos de um sistema *blockchain*. Essas transações são registradas da seguinte forma: primeiro as transações são coletadas e transfomadas em blocos, esses blocos têm um tamanho máximo para garantir a sua eficiência e escalabilidade da rede. Antes dessas transações serem inseridas no bloco, elas passam por uma validação realizada por uma rede de nós dessa *blockchain*, dessa forma garantindo sua autênticidade para realizar a transação. A próxima etapa é o consenso, o qual também faz parte da validação das transações e seu objetivo é que os nós da rede cheguem em um acordo sobre quais transações devem ser incluídas no bloco. Essa etapa será abordada com mais detalhes no próximo parágrafo. Após essas etapas serem realizadas, a transação é adicionada a cadeia de blocos, assim se tornando permanente no registro. Para cada bloco inserido no sistema *blockchain*, existe um código *hash* individual que faz referência ao seu bloco antecessor. Essa

referência faz esses blocos se interligarem, criando assim uma cadeia de blocos. Nesse sentido, qualquer alteração feita em um bloco causa alterações em todos os outros subsequentes, tornando assim o sistema seguro contra alterações.

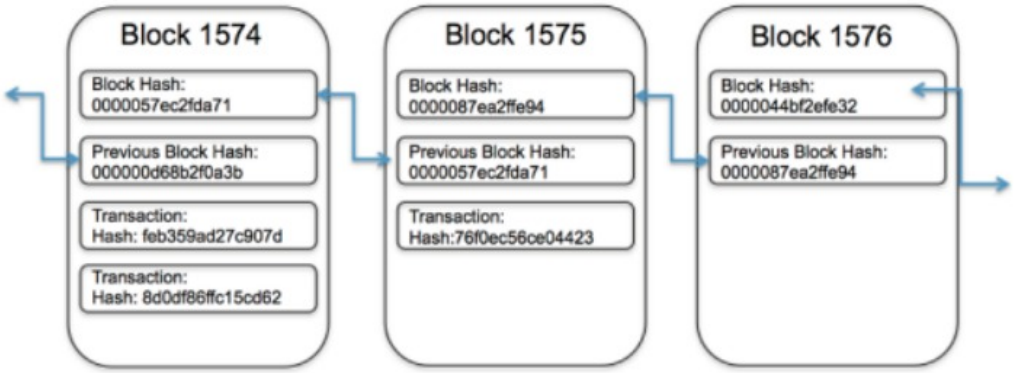


Figura 2: ilustração de um registro de transações em cadeia de blocos (*blockchain*).

Fonte: Cagliari, Silva e Santos (2022)p.5.

Para que um novo bloco seja acrescentado à cadeia, a maioria dos nós na rede deve validar tanto o novo bloco quanto as transações associadas a ele. Esse processo é conhecido como **mecanismo de consenso**. Este mecanismo é um procedimento em que todos os nós na rede da *blockchain* precisam chegar a um acordo sobre a mesma informação. Isso assegura que a última modificação seja incorporada de maneira apropriada, evitando assim potenciais alterações maliciosas (LIN; LIAO, 2017).

Dentre os dois principais mecanismos de consenso, o *proof of work* (PoW) e o *proof of stake* (PoS), podemos observar que o PoW requer que todos os nós na rede da blockchain tenham uma cópia do registro geral (livro-razão) para resolver um quebra-cabeça complexo baseado na última versão do registro. As máquinas que possuem cópias idênticas formam grupos. O primeiro grupo a resolver o quebra-cabeça vence, e todas as máquinas consideradas perdedoras atualizam seus registros para que fiquem iguais aos do grupo vencedor (LAURENCE, 2023).

Por outro lado, o PoS representa uma alternativa mais econômica em termos de consumo de energia e uma abordagem mais eficiente em comparação com o PoW. Nesse processo, a seleção das máquinas é determinada pela quantidade de moedas que um minerador possui. Acredita-se que quanto mais moedas um participante detém, menos provável ele será de se envolver em atividades maliciosas ou ataques à rede (ZHENG *et al.*, 2017).

Existe outro mecanismo de consenso não muito utilizado chamado *Delegated Proof os Stake* (DPoS). Esse mecanismo segue o modelo do PoS com uma velocidade maior, contudo apresenta diversas falhas de segurança por ter uma menor descentralização. O DPoS funciona a partir da escolha de uma pessoa que foi eleita por outros usuários, logo ela terá poder sobre esses eleitores. Contudo, a pessoa eleita poderá abusar do poder, uma vez que o mesmo tenha a responsabilidade de validar a cadeia de blocos. Caso o representante do sistema apresentar um desempenho inferior ou deturpar seus integrantes, os usuários poderam retirar essa posse e realizar uma outra votação (SAAD; RADZI, 2020).

A **Figura 3** apresenta uma tabela de comparação entre alguns dos principais mecanismos de consenso. Na comparação são apresentados três mecanismos e suas respectivas características: seu consumo de energia, transações por segundos o qual representa o seu desempenho de processamento, a taxas de transação que equivalem aos preços que os usuários pagam para enviar uma transação em um sistema *blockchain*, sua estrutura e um exemplo real de uso de cada mecanismo.

	Proof of Work (PoW)	**Proof of Stake (PoS)**	**Proof of Stake Dedicada (DPoS)**
Consumo de Energia	Alto	Baixo	Muito baixo
Transações por segundo	7 a 30	30 a 173	25 a 2500
Taxas de transação	Alto	Baixo	Baixo
Estrutura	Descentralizada	Descentralizada	Centralizada
Exemplo	Bitcoin	Dash / Ethereum	Bitshares

Fonte : imagem adaptada do trabalho (DUBROWSKY, 2019)p.40.

Figura 3: comparação entre os mecanismos de consenso.

A tecnologia *Blockchain* é composta por uma rede *Peer-to-Peer* (P2P). Essas redes são comumente ligadas à disseminação de arquivos de mídia, incluindo áudio e vídeo. Além disso, essa tecnologia também encontra aplicação na distribuição de volumes significativos de informações, como atualizações de software, serviços de backup e sincronização de dados entre múltiplos servidores. Essa tecnologia ganhou popularidade para esses propósitos, principalmente devido à sua capacidade de acomodar um grande número de usuários (*Peers*) e ao seu custo operacional relativamente baixo em comparação com alternativas, como o modelo convencional baseado em cliente-servidor.

Atualmente, a tecnologia *blockchain* pode ser classificada em três categorias distintas, cada uma delas com características e abordagens específicas: *blockchain* pública

sem permissão, *blockchain* pública permissionada e *blockchain* privada permissionada. As principais diferenças entre essas categorias residem nas maneiras pelas quais os usuários podem interagir e participar na rede (PEDERSEN *et al.*, 2019). Os três tipos de *blockchain* são:

- A *blockchain* pública sem permissão é uma rede aberta que permite a participação de qualquer pessoa (exemplos incluem *Bitcoin e Ethereum*). Com esse tipo de *blockchain*, todos os usuários podem ler, escrever e verificar transações. Esse tipo de *blockchain* pode substituir o papel de um terceiro confiável. A confiança é construída entre pares na rede, porque todos eles têm que respeitar o estabelecido mecanismo de consenso. Os mais populares mecanismos de consenso são o PoW e o PoS;

- A *blockchain* pública permissionada é uma rede fechada, em que apenas nós verificados e confiáveis podem participar; alguns exemplos de redes que adotam esse tipo são a *Ripple, Multichain, Eris e Hyperledger Fabric*. Esse tipo também é chamado de "*blockchain híbrida*", porque todos os participantes podem visualizar os dados, mas apenas usuários autorizados podem validar as transações. Os usuários são autorizados através de um consenso de rede depois de fornecer a prova necessária de elegibilidade;

- A *blockchain* privada permissionada é uma rede fechada que permite apenas usuários autorizados a ler, enviar e validar as transações. Alguns exemplos destas redes incluem a *Hyperledger Fabric* e Corda. As transações são verificadas ou o consenso da *blockchain* é determinado dentro de uma organização. Geralmente é utilizado um protocolo de *Byzantine Fault Tolerance* (BFT), o qual requer uma certa porcentagem de nós previamente verificados para confirmar as transações.

A **Figura 4** apresenta uma tabela comparando as duas principais categorias de um sistema *blockchain*, sendo elas a pública e a privada. Essa tabela apresenta quais são as propriedades específicas de cada *blockchain*, sendo elas a velocidade, a segurança, privacidade, sua estrutura e seu consumo de energia.

	Blockchain Privada	*Blockchain* Pública
Velocidade	Mais rápidos e escaláveis. Existência de participantes autorizados resulta em tempos significativamente menores na obtenção de um consenso.	Mais lento, pois a obtenção de consenso sobre o estado das transações leva muito mais tempo. Há limites com relação a quantidade de transações por bloco.
Segurança	Maior propensão a manipulação. A integridade da rede depende da credibilidade de nós autorizados. Atores externos precisam confiar sem ter controle sobre o processo de verificação.	Transações são publicamente transparentes e imutáveis. Dados não podem ser adulterados ou alterados. Com mais nós na rede é mais difícil para um ator mal intencionando atacar a rede
Privacidade	Facilidade para restringir acesso a dados ou a determinadas funções.	Dificuldade para manter dados em sigilo, pois todos os dados relacionados e transações estão abertos para serem verificados pelo público
Centralização / Descentralização	Centralizada	Descentralizada
Consumo energia	Menor consumo	Maior consumo. O algoritmo POW consome grande quantidade de recursos elétricos para funcionar, levantando preocupações com o meio ambiente.

Fonte : Imagem adaptada do trabalho (DUBROWSKY, 2019)p.37.

Figura 4: comparação entre blockchain privada e pública.

A tecnologia *blockchain* vem ganhando cada vez mais notoriedade em organizações de diversas instituições de ensino superior, prometendo resolver problemas relacionados a certificados acadêmicos, como a padronização, emissão e autenticação de informações por parte dos interessados. Ela é ideal para a proteção e compartilhamento de informações para qualquer pessoa ou instituição e permite validar imediatamente os dados sem precisar de um intermediário (SMOLENSKI; HUGHES, 2016).

2.3 *Blockchain* para segurança em certificações acadêmicas digitais

Com o avanço tecnológico, as certificações acadêmicas digitais surgiram para solucionar os problemas de certificados físicos. Essas dificuldades foram discutidas com mais detalhes na **Seção 2.1**. Por outro lado, há um ponto em específico que ainda continua em aberto para a maioria das instituições acadêmicas: a confiança dos agentes envolvidos. Geralmente para uma pessoa física ou jurídica emitir e validar um certificado digital, ela precisa obrigatoriamente de uma entidade central que faça a autenticação desse documento, dessa forma o tornando válido.

Esse processo de validação por instituições centrais pode ter um potêncial problema, pois em um sistema centralizado, somente uma autoridade certificadora pode

emitir e gerenciar os certificados digitais para um usuário. Caso a AC for manipulada por pessoas com má intencionadas, poderá ocorrer uma falha de segurança no processo, e, por consequência, alguma pessoa será prejudicada. Podemos citar um caso real que aconteceu no final de 2022, no qual foi constatada a existência de uma fraude na emissão de certificados digitais por parte de uma empresa certificadora. Segundo o Tribunal Regional do Trabalho 1ª Região do Rio de Janeiro, os certificados digitais eram alvarás utilizados por juízes para o pagamento de processos jurídicos. De acordo com a mídia, a fraude pode ultrapassar o valor de 4 milhões de reais (GUIMARãES, 2023).

Em virtude desse problema, a solução mais viável é a implementação da tecnologia *blockchain*. Essa tecnologia discutida na **Seção 2.2** se enquadra em um sistema descentralizado, desse modo não necessitando de uma terceira entidade nos registros. Esse modelo descentralizado resolve o problema das entidades certificadoras uma vez que todos os nós presentes no sistema são interligados entre si e precisam entrar em consenso para que seja feito qualquer modificação nos registros.

Em um cenário acadêmico, com a utilização do *blockchain*, os certificados digitais poderiam ser verificados a qualquer momento, independente de algum imprevisto com a universidade emissora, até mesmo se ela não existir mais. Esse cenário não é visto em sistemas centralizados, pois se houver qualquer imprevisto externo o acesso ao certificado pode ser comprometido permanentemente, mesmo que ele seja válido (HAUCK, 2023).

A primeira implementação da *blockchain* dentro de uma instituição acadêmica foi feita em 2014 pela Universidade de Nicosia (UNIC). Essa universidade introduziu a *blockchain* do *Bitcoin* para realizar a emissão e verificação dos certificados acadêmicos, sendo assim foi a primeira a explorar o potencial da tecnologia junto à educação (University of Nicosia, 2020). Já em 2015, a universidade *Massachusetts Institute of Technology* (MIT), juntamente com seu departamento de pesquisas MIT Media Lab, desenvolveram a plataforma *Blockcerts* com o intuito de estudar a implementação do *blockchain* em intituições acadêmicas. Em 2017 com a colaboração da empresa *Learning Machine*, o MIT criou um projeto teste com o código provido do *Blockcerts* para a emissão dos diplomas de dois cursos que a universidade oferecia. Com o *feedback* de seus alunos, eles foram capazes de identificar quais os principais pontos para o desenvolvimento em larga escala (BLOCKCERTS, 2016). Em 2016, o país de Malta foi um dos pioneiros no desenvolvimento do *blockchain* para a adoção de credenciais

acadêmicas e certificações profissionais. O intuito do governo do país é expandir em nível nacional esse modelo voltado para a educação, criando uma escalabilidade e flexibilidade maiores (GRECH, 2017).

No Brasil, a adoção da tecnologia *blockchain* para emissão e registro de certificados acadêmicos foi implementada pela primeira vez no ínicio de 2019 pela Universidade Federal da Paraíba (UFPB). Segundo a UFPB, a plataforma desenvolvida vai ao encontro de uma demanda crescente nos meios universitários por mais segurança na expedição de diplomas e outros documentos que certificam a formação em nível superior. Diante do crescente registro de casos de falsificação de diplomas em várias regiões do país, o próprio Ministério da Educação instituiu o diploma em formato digital, conforme Portaria nº 330, de 5 de abril de 2018 (BRASIL, 2019). No mesmo ano, a Universidade Federal de Santa Catarina (UFSC) também implementou o *blockchain* para a emissão de diplomas. Além disso, o projeto de emissão de certificados da UFSC tem uma visão de futuro, onde todo o histórico acadêmico do aluno poderia ser registrado na rede, desde o ensino fundamental até o ensino superior. Ainda que algumas soluções tenham ancorado registros na cadeia do *Bitcoin*, parte da comunidade acadêmica passou a preferir arranjos com contratos inteligentes no *Ethereum*, que permitem codificar regras de emissão, verificação e revogação diretamente no contrato e disponibilizar trilhas de auditoria no próprio registro (PALMA *et al.*, 2019). A intenção é reduzir a dependência de um operador central e a discricionariedade técnica na validação, sem transferir a responsabilidade jurídica pelo ato de emitir, que permanece com a universidade.[3]

2.4 *Ethereum*

Entre as diversas aplicações do *blockchain*, o *Ethereum* se destacou como uma das tecnologias que trouxe maior notoriedade para a rede *blockchain*. O *Ethereum* se baseia em máquinas virtuais descentralizadas, denominadas como *Ethereum Virtual machines* (EVM). Nesta plataforma é possível criar programas autoexecutáveis que são realizadas quando as condições dos termos pré-programados são atendidas. Esses programas são nomeados como *contratos inteligentes* (MASCARENHAS; VIEIRA; ZIVIANI, 2018).

[3]"Responsabilidade", aqui, refere-se à operação e governança técnica do processo de validação. A responsabilidade legal pela emissão do documento acadêmico continua sendo da instituição emissora, conforme normas aplicáveis.

O conceito de contratos inteligentes surgiu em 1996 pelo criptógrafo e advogado Nick Szabo. O artigo publicado denomina-se *Smart Contracts: Building Blocks for Digital Free Markets* e nele é encontrado todo o conteúdo sobre os contratos inteligentes e quais são as soluções que poderiam ser resolvidas utilizando os contratos inteligentes junto com a criptografia. O artigo também aponta quais as vantagens do contratos inteligentes comparados aos contratos feitos da forma tradicional (SZABO, 1996; LAMOUNIER, 2018).

A **Figura 5** apresenta uma comparação entre os contratos tradicionais feitos de forma escrita e os contratos inteligentes. Nessa figura são destacados os principais pontos entre eles como: a criação mais rápida por parte dos contratos inteligentes, a automatização para a remessa dos contratos inteligentes, a não necessidade de um terceiro indivíduo para manter e proteger os ativos no contrato inteligente, conhecido como custodiante, os custos são reduzidos nos contratos inteligentes, assinaturas presenciais e quaisquer intermediários não são mais necessários em contratos inteligentes.

Fonte : (JURIDOC, 2022)

Figura 5: comparação entre contratos inteligentes e tradicionais.

A plataforma *Ethereum* foi implementada em 2014 por três pessoas: Vitalik Buterin, Gavin Wood e Jeffrey Wilcke. A partir disso se iniciou a segunda geração da *blockchain*. Junto com a plataforma *Ethereum*, foi criado a moeda virtual *Ether* ou identificada também como "ETH", sendo capaz de ativar e executar os contratos inteligentes (PEREIRA, 2018).

Na rede *Ethereum*, os mineradores que realizam as operações de validações também são recompensados pela quantidade de computação que eles oferecem. O nome dessa taxa que os mineradores do *Ethereum* recebem para executar os contratos inteligentes se denominam *Gwei*. Essa palavra é uma abreviação de *Giga-wei* e equivale a 1 bilhão de *wei*. Wei é a menor unidade do *ether*, criptomoeda nativa do *Ethereum*. Essa unidade de medida é determinada pelo usuário, definindo o quanto o mesmo está disposto a pagar pela transação em *Gwei* por unidade por *Gas*, isso pode afetar a prioridade e velocidade do processamento da transação. O *Gas* também é uma unidade de medida, porém ela é usada para medir o consumo de recursos computacionais durante uma execução de uma operação no *Ethereum*. A taxa total que o usuário vai pagar ao minerador é determinada pelo número de unidades de *gas* multiplicado pela taxa em *Gwei* que o usuário está disposto a pagar (AKIAU *et al.*, 2023).

Como dito na **Seção 2.3** o laboratório do *MIT* foi uma das primeiras instituições acadêmicas a implementar a criação, emissão e verificação de certificados acadêmicos usando a *blockchain* do *Bitcoin*, denominado *Blockcerts*. A *TrueRec* da empresa *SAP* introduziu uma funcionalidade adicional de armazenamento de certificados na *blockchain*, ao contrário da *BlockCerts*, a empresa escolheu a blockchain *Ethereum* para essa finalidade. Em ambos os casos, os estudantes possuem uma carteira digital na qual podem guardar seus certificados e compartilhá-los diretamente com seus empregadores. Os empregadores, por sua vez, podem inserir o código recebido na plataforma atual para comparar os resultados apresentados com os certificados recebidos (CAGLIARI; SILVA; SANTOS, 2022).

2.5 Bibliotecas e *frameworks*

Utilizou-se a linguagem de programação *Solidity* (SOLIDITY, 2025), destinada a contratos inteligentes na máquina virtual compatível com *Ethereum*, com o objetivo de codificar e fazer cumprir regras de negócio na rede. Empregou-se o *Truffle* (TRUFFLE, 2025) para compilar contratos, gerar artefatos e conduzir migrações, padronizando o ciclo de desenvolvimento e *deploy*. Recorreu-se ao *Ganache* (TRUFFLE SUITE, 2025) como *blockchain* local compatível com *Ethereum*, oferecendo rede de testes com contas e Chamadas Remotas de Procedimento (RPC) para desenvolvimento. Adotou-se o *React* (META OPEN SOURCE, 2025), biblioteca de interface baseada em componentes, para construir aplicações *web* reativas. Organizou-se a na-

vegação com *React Router*(REMIX SOFTWARE, 2025), solução de rotas para aplicações de página única. Estabeleceu-se a integração com a *blockchain* por meio do *web3.js* (ETHEREUM FOUNDATION, 2025), possibilitando leitura de estados, envio de transações e consumo de eventos. Centralizou-se a gestão de contas e assinaturas no *Metamask* (METAMASK, 2025), carteira digital e provedor de assinaturas no navegador. Executaram-se *scripts* e administraram-se dependências com *Node.js* e *npm*, ambiente de execução *JavaScript* e gerenciador de pacotes (OPENJS FOUNDATION, 2025; NPM, INC., 2025). Implementou-se a geração de imagens no navegador, técnica para produzir artefatos visuais a partir de dados e criar documentos compartilháveis (BUBENSHCHYKOV; CONTRIBUTORS, 2025). Incorporou-se o código QR (ISO/IEC, 2015), representação visual legível por câmera, para facilitar acesso e verificação em dispositivos móveis.

Capítulo 3

Sistema Interface de gestão de certificação acadêmica

Este capítulo descreve o desenvolvimento do sistema, da modelagem dos contratos inteligentes à implementação da interface *web* e da integração com a rede compatível com *Ethereum*. Apresentam-se as decisões de projeto, a arquitetura adotada e os principais componentes que tornam possível emitir, revogar e verificar certificações acadêmicas digitais com rastreabilidade e controle de acesso.

3.1 Visão geral funcional do sistema

Perfis de usuário:

- **Autoridade emissora:** órgão ou instituição habilitada a credenciar instituições e definir quem pode emitir/assinar;
- **Instituição emissora:** unidade acadêmica que emite e, quando necessário, revoga certificações;
- **Titular (estudante/egresso):** pessoa à qual a certificação se refere; pode consultar e compartilhar sua certificação;
- **Verificador externo:** terceiro que checa a autenticidade/integraidade de uma certificação apresentada.

Objetos e dados principais:

- **Autoridade:** identificação da autoridade, endereço na rede, *status* (ativa/inativa) e eventos de credenciamento;

- **Currículo:** registra o histórico acadêmico do aluno, incluindo identificadores, disciplinas concluídas e *hashes* de metadados *off-chain*, que armazenam informações adicionais fora da *blockchain* para reduzir custos e preservar integridade;
- **Diploma/Certificação:** identificador único, *hash* do documento, emissor, titular, carimbo temporal, validade e estado (ativo/revogado).

Funcionalidades:

- **Credenciamento**: a autoridade habilita/desabilita instituições emissoras;
- **Registro de currículo**: a instituição cadastra/atualiza informações e *hashes* associados ao titular;
- **Emissão**: a instituição emite a certificação vinculando *hash*, titular e metadados essenciais;
- **Verificação**: qualquer parte consulta o registro e compara o *hash* do documento apresentado com o armazenado na rede;
- **Revogação**: a instituição altera o estado da certificação para revogada, preservando registro imutável;
- **Compartilhamento**: geração de imagem do diploma com o código *QR* que aponta ao verificador e facilita a conferência pública.

Regras de acesso e auditoria:

- Modificadores de acesso restringem operações críticas (ex.: `onlyAuthority`, `onlyIssuer`, `onlyOwner`);
- Eventos registram credenciamento, emissão, verificação e revogação, formando registro rastreável;
- Separação *on/off-chain*: informações sensíveis, como dados pessoais e documentos completos, são armazenadas fora da *blockchain* (*off-chain*) e dentro da *blockchain* (*on-chain*) ficam apenas identificadores, estados de certificados e hashes que garantem integridade e verificabilidade.

Primeiro, detalham-se os contratos escritos em *Solidity* e a separação de responsabilidades entre *Autoridade*, *Currículo* e *Diploma*, com suas estruturas de dados, modificadores de acesso e eventos. Em seguida, apresentam-se o ambiente de compilação e migração com *Truffle* e a rede local com *Ganache*, que sustentam o ciclo de desenvolvimento. A camada cliente é implementada em *React* com *React Router* para navegação. Na sequência, descreve-se a integração com a *blockchain* por meio do *Web3.js* e o gerenciamento de contas/assinaturas no *MetaMask*. Por fim, mostra-se a geração da imagem do diploma e o uso de *QR* para verificação pública, compondo um artefato legível fora da cadeia conectado ao registro *on-chain*.

3.2 Solidity e contratos

Linguagem de programação voltada a contratos inteligentes na máquina virtual *Ethereum*. Define estruturas de dados, modificadores de acesso, eventos e funções que regem emissão, revogação e verificação de certificados.

A **Listagem 3.2.1** mostra o ciclo de vida do certificado de uma instituição. A função *adicionarCertificadoInstituicao* grava o *hash* e a validade, marca o registro como ativo e não revogado e dispara um evento que registra a emissão com a data do bloco.

Listagem 3.2.1: trecho do contrato "*Autoridade*: Emissão".

```
1  function adicionarCertificadoInstituicao(address instituicao, bytes32
       hashCertificado, uint validade)
2      public somenteDono
3  {
4      require(instituicao != address(0), "Endereco da instituicao invalido
           ");
5      certificados[instituicao].hash = hashCertificado;
6      certificados[instituicao].validade = validade;
7      certificados[instituicao].revogado = false;
8      certificados[instituicao].registrado = true;
9
10     emit CertificadoEmitido(dono, instituicao, block.timestamp);
11 }
```

A **Listagem 3.2.2** mostra o evento *CertificadoRevogado* que define o formato do *log* de revogação com emissor, alvo e data. A função *revogarCertificadoInstituicao* altera o estado para revogado, inclui o endereço em uma lista de revogados e emite o evento de revogação.

Listagem 3.2.2: trecho do contrato "*Autoridade*: revogação".

```
event CertificadoRevogado(address de, address para, uint data);

function revogarCertificadoInstituicao(address enderecoInstituicao)
    public somenteDono
{
    require(
        enderecoInstituicao != address(0) &&
        certificados[enderecoInstituicao].registrado,
        "Endereco do certificado invalido ou nao registrado"
    );

    certificados[enderecoInstituicao].revogado = true;
    revogados.push(enderecoInstituicao);

    emit CertificadoRevogado(dono, enderecoInstituicao, block.timestamp)
        ;
}
```

A **Listagem 3.2.3** apresenta função *verificarCertificadoInstituicao* que confirma se existe registro, se ele não foi revogado e se a sua validade ainda está em vigor, retornando falso quando houver revogação ou vencimento.

Listagem 3.2.3: trecho do contrato "*Autoridade*: verificação".

```
function verificarCertificadoInstituicao(address instituicao)
    public view returns (bool)
{
    require(
        instituicao != address(0) &&
        certificados[instituicao].registrado,
        "Instituicao invalida ou nao registrada"
    );

```

```
10      if (
11          certificados[instituicao].revogado ||
12          certificados[instituicao].validade < block.timestamp
13      ) return false;
14
15      return true;
```

A **Listagem 3.2.4** organiza cadastro e progresso acadêmico do aluno. A função *registrarEstudante* recebe o endereço do aluno e os dados básicos, rejeita endereços nulos ou já registrados, grava nome e curso, zera o contador de concluídas, marca o aluno como registrado e indica que o diploma ainda não está pronto. Também guarda no mapa qual instituição fez o registro e emite o evento *EstudanteRegistrado* com a data do bloco.

Listagem 3.2.4: trecho do contrato "*Curriculo*: registro de estudante".

```
1  function registrarEstudante(address estudante, string memory nome,
       string memory curso)
2      public
3      somenteInstituicaoAutorizada
4  {
5      require(
6          estudante != address(0) && !estudantes[estudante].registrado,
7          "Endereco do estudante invalido ou ja registrado"
8      );
9
10     estudantes[estudante].nome = nome;
11     estudantes[estudante].registrado = true;
12     estudantes[estudante].concluidas = 0;
13     estudantes[estudante].curso = curso;
14     diplomaPronto[estudante] = false;
15
16     // guarda quem registrou
17     instituicaoDoAlunoMap[estudante] = msg.sender;
18
19     emit EstudanteRegistrado(msg.sender, estudante, block.timestamp);
20 }
```

A **Listagem 3.2.5** apresenta a função *registrarDisciplina* que cria uma disciplina identificada por um código, verifica se ainda não existe, marca como registrada e salva nome e carga horária.

Listagem 3.2.5: trecho do contrato "*Curriculo*: registro de disciplinas".

```
1
2  function registrarDisciplina(bytes32 id, string memory nome, uint
       cargaHoraria)
3      public
4      somenteInstituicaoAutorizada
5  {
6      require(!disciplinas[id].registrada, "Disciplina ja registrada");
7      disciplinas[id].registrada = true;
8      disciplinas[id].nome = nome;
9      disciplinas[id].duracao = cargaHoraria;
10 }
```

A **Listagem 3.2.6** mostra a função *disciplinaConcluida* do contrato Currículo registra ou remove a conclusão de uma disciplina de um aluno, atualizando seu histórico acadêmico. Apenas a instituição que registrou o aluno pode executar essa ação e apenas instituições autorizadas podem chamar a função. O contrato verifica se o aluno e a disciplina estão cadastrados antes de atualizar o status. Sempre que o número de disciplinas concluídas atinge a carga mínima, o aluno é marcado como apto a receber o diploma e o evento correspondente é registrado.

Listagem 3.2.6: trecho do contrato "*Curriculo*: registro de disciplinas concluídas".

```
1
2  // somente a instituicao que registrou o aluno pode marcar/retirar
       conclusao
3  function disciplinaConcluida(bytes32 disciplina, bool status, address
       estudante)
4      public
5      somenteInstituicaoAutorizada
6      somenteInstituicaoDoAluno(estudante)
7  {
8      require(
9              estudante != address(0) &&
10             estudantes[estudante].registrado &&
```

```
            disciplinas[disciplina].registrada,
            "Estudante ou disciplina invalidos ou nao registrados"
        );

        bool anterior = estudantes[estudante].creditos[disciplina].
            concluida;
        estudantes[estudante].creditos[disciplina].concluida = status;

        if (!anterior && status) {
            estudantes[estudante].concluidas += 1;
        } else if (anterior && !status && estudantes[estudante].
            concluidas > 0) {
            estudantes[estudante].concluidas -= 1;
        }

        if (estudantes[estudante].concluidas >= cargaMinima) {
            diplomaPronto[estudante] = true;
            emit DiplomaPronto(estudante);
        } else {
            diplomaPronto[estudante] = false;
        }
    }
}
```

Este trecho da **Listagem 3.2.7** define emissão a do diploma. A função *emitirDiploma* só prossegue quando o diploma ainda não foi emitido, quando o aluno pertence à mesma instituição concedente e quando o Currículo informa que o aluno está apto a graduar. Ao emitir, grava a data com o tempo do bloco, marca o status como verdadeiro e dispara o evento de diploma emitido com dados do emissor, do receptor e da data.

Listagem 3.2.7: trecho do contrato "*Diploma*: emissão de diplomas".

```
function emitirDiploma() public somenteConcedenteCertificada {
        require(!diploma.status, "Ja emitido");
        require(curriculo.instituicaoDoAluno(dono) == concedente, "Aluno
             nao pertence a esta instituicao");
        require(curriculo.alunoAptoAGraduar(dono), "O estudante nao
```

```
            concluiu todas as disciplinas");

        diploma.data = block.timestamp;
        diploma.status = true;
        emit DiplomaEmitido(diploma.emissor, diploma.receptor, diploma.
            data);
    }
```

A **Listagem 3.2.8** apresenta a função *revogarDiploma* exige que o diploma esteja ativo e repete a verificação de vínculo do aluno com a instituição. Em seguida, marca o status como falso e registra o evento de diploma revogado com o endereço do concedente e a data do bloco. A função *diplomaValido* expõe uma leitura simples do status, retornando verdadeiro quando o diploma está ativo e falso quando já foi revogado ou ainda não foi emitido.

Listagem 3.2.8: trecho do contrato "*Diploma*: revogação de diplomas".

```

function revogarDiploma() public somenteConcedenteCertificada {
        require(diploma.status, "Nao emitido ou ja revogado");
        require(curriculo.instituicaoDoAluno(dono) == concedente, "Aluno
             nao pertence a esta instituicao");

        diploma.status = false;
        emit DiplomaRevogado(concedente, block.timestamp);
    }
    function diplomaValido() public view returns (bool) {
        return diploma.status;
    }
```

3.3 Truffle

Truffle (TRUFFLE, 2025) atua como ponte entre os contratos escritos em *Solidity* e a interface do usuário. Ele compila os contratos, executa as migrações e publica os endereços dos contratos implantados. Durante a compilação o *Truffle* gera arquivos

JSON chamados *artifacts* que contêm duas informações fundamentais: a *ABI*, que significa *Application Binary Interface* e descreve em formato *JSON* as funções, eventos e tipos do contrato permitindo ao *frontend* codificar chamadas e decodificar respostas; e o *bytecode*, que é o código binário resultante da compilação e que é enviado para a máquina virtual *Ethereum* quando o contrato é implantado. Esses *artifacts* são consumidos pelo *frontend* para possibilitar a comunicação com os contratos. A *blockchain* local *Ganache* é usada como rede de desenvolvimento para testes e o *contracts-build-directory* foi configurado para salvar os *artifacts* em *src/abi* do *frontend*. Nas migrações são implantados os contratos Autoridade e Currículo. O contrato Diploma é criado dinamicamente pela interface sempre que necessário.

A **Listagem 3.3.1** mostra o arquivo de configuração do *Truffle*. Ele define onde os artefatos da compilação serão gravados, qual rede local será usada e qual versão do compilador *Solidity* será aplicada. A diretiva *contracts-build-directory* envia os arquivos *JSON* para a pasta do *front end* em *src/abi*, permitindo que a aplicação leia o *ABI* e os endereços logo após a migração. Entre as linhas 6 e 10, o bloco de redes com o nome *development* aponta para o *Ganache* usando o *host* 127.0.0.1, a *porta* 7545 e o *network id* 1337, o que faz com que as migrações implantem os contratos nessa rede local. Entre as linhas 12 e 14 a seção de compiladores fixa a versão do *solc* como 0.8.0, que deve estar compatível com a versão declarada nos contratos. Para validar o ambiente, basta conferir se o caminho de *build* existe, se a porta e o *network id* batem com o *Ganache* e se a versão do compilador condiz com a *pragma* dos contratos.

Listagem 3.3.1: arquivo do *Truffle-config.js*.

```
1  const path = require("path");
2  module.exports = {
3    // envia os JSONs direto pro front
4    contracts_build_directory: path.join(__dirname, "../../my-dapp-1/src/
       abi"),
5    networks: {
6      development: {
7        host: "127.0.0.1",
8        port: 7545, // porta do Ganache GUI
9        network_id: "1337" // Network ID
10     },
11   },
12   compilers: {
```

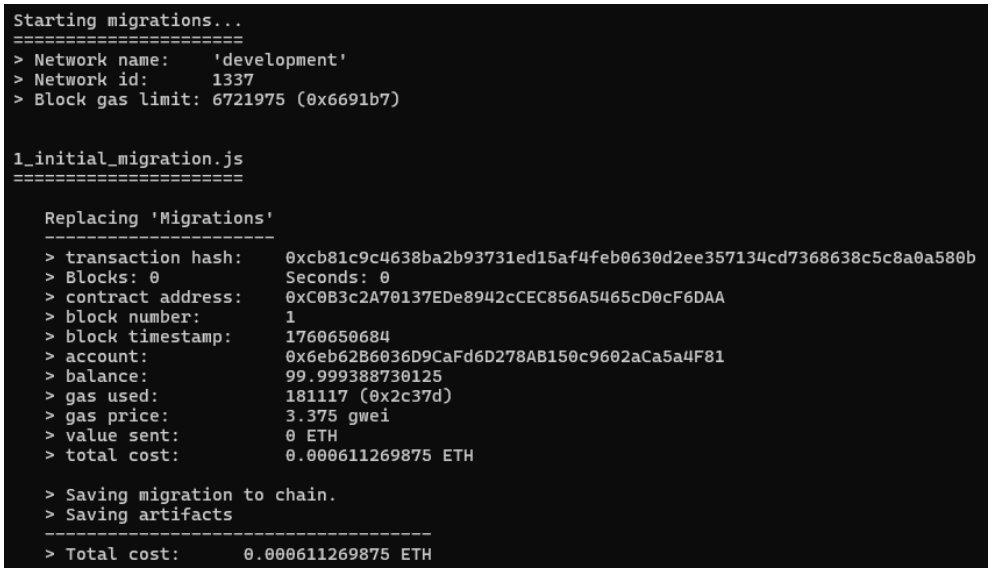
```
13    solc: { version: "0.8.0" } // versão dos meus contratos
14  }
15 };
```

A **Listagem 3.3.2** representa o arquivo de migração que carrega os artefatos dos contratos e executa a implantação na rede de desenvolvimento. Primeiro solicita o contrato Autoridade, envia a transação de *deploy* e aguarda a conclusão para obter a instância implantada. Em seguida, escolhe como instituição inicial a primeira conta disponibilizada pelo *Ganache*. Depois executa o *deploy* do contrato Currículo informando três parâmetros na ordem esperada, a quantidade mínima de disciplinas a cumprir, aqui definida como um, o endereço do contrato Autoridade recém-implantado e o endereço da instituição inicial. O contrato Diploma é citado no início do arquivo, mas não é implantado nesta etapa, pois sua criação ocorre sob demanda durante o uso da aplicação.

Listagem 3.3.2: arquivo do *Deploy-Contracts.js*.

```
1  const Autoridade = artifacts.require("Autoridade");
2  const Curriculo = artifacts.require("Curriculo");
3  const Diploma = artifacts.require("Diploma");
4
5  module.exports = async function (deployer, network, accounts) {
6    // Deploy Autoridade
7    await deployer.deploy(Autoridade);
8    const autoridade = await Autoridade.deployed();
9    // endereço válido do ganache, prioriza a primeira conta do ganache
10   const instituicaoInicial = accounts[0];
11
12   // Deploy Curriculo apontando para essa Autoridade
13   await deployer.deploy(
14     Curriculo,
15     1, // DisciplinasACumprir
16     autoridade.address, // ContratoAutoridade
17     instituicaoInicial // instituicao
18   );
19 };
```

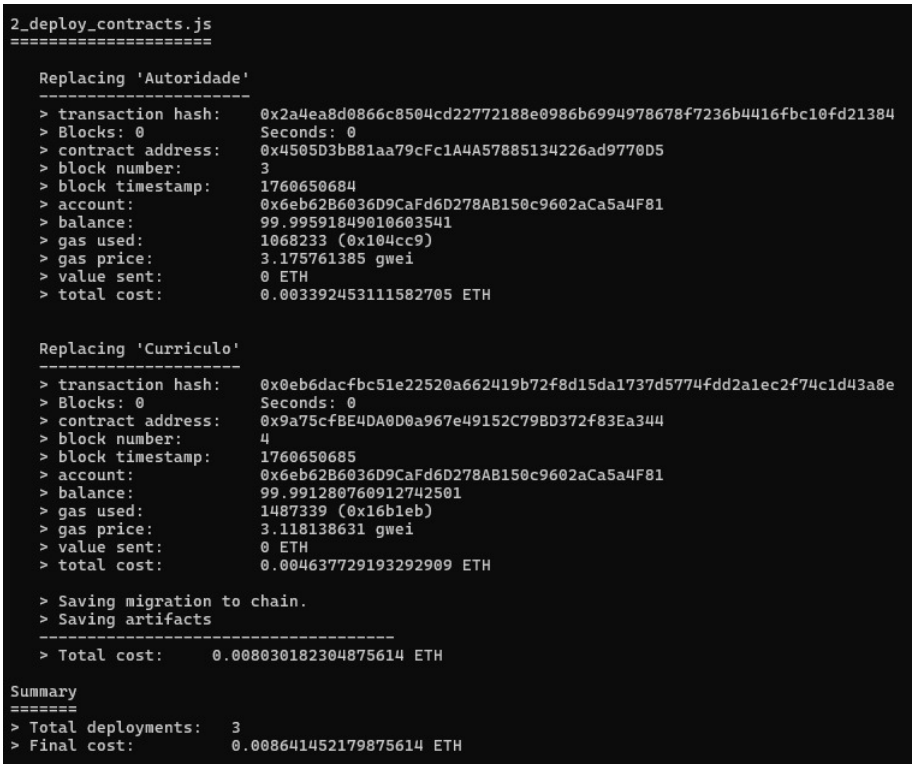
A **Figura 6** mostra o registro da execução completa das migrações no ambiente de desenvolvimento. A primeira linha informa que as migrações começaram na rede chamada *development* com *network id* igual a 1337 e com o limite de gás do bloco. Em seguida aparece a etapa inicial que implanta o contrato *Migrations*. O terminal mostra o *hash* da transação, o número do bloco, o horário do bloco, a conta usada, o gás consumido, o preço do gás, o valor enviado que é zero e o custo total. Ao final desta etapa o *Truffle* salva a migração e grava os artefatos.



Fonte: imagem do autor.

Figura 6: execução da migração do ambiente de desenvolvimento.

Na **Figura 7** ocorre a sequência a migração principal. Primeiro é implantado o contrato Autoridade. O terminal mostra o *hash* da transação, o endereço do contrato gerado, o número do bloco e os custos. Depois é implantado o contrato Currículo, também com *hash*, endereço do contrato, bloco e custos. A migração termina com a confirmação de salvamento e com os artefatos escritos na pasta configurada.



Fonte: imagem do autor.

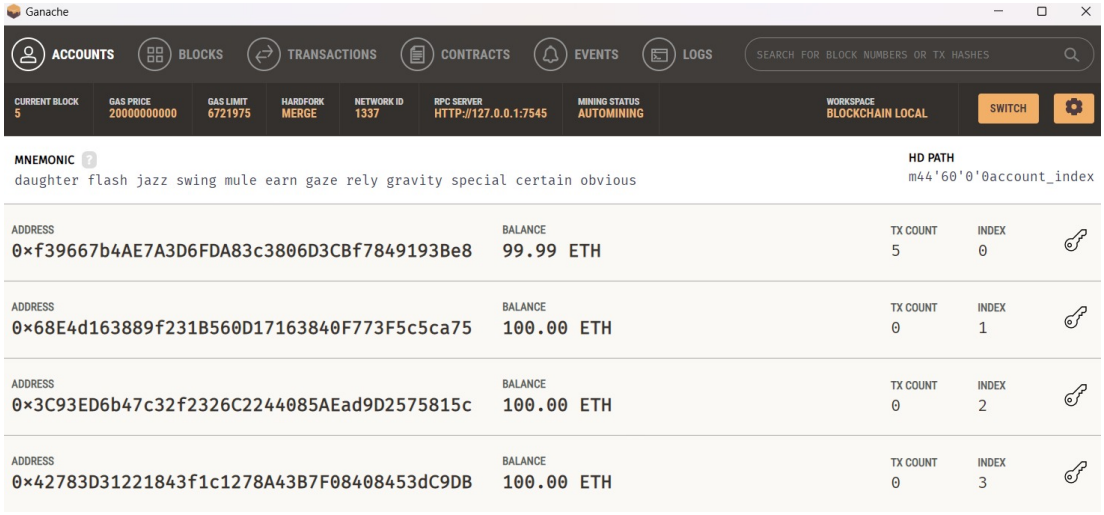
Figura 7: execução da migração dos contratos de autoridade e currículo.

3.4 Ganache

Ganache (TRUFFLE SUITE, 2025) é uma rede *blockchain* local voltada para desenvolvimento e testes. Ele simula um nó compatível com a máquina virtual de *Ethereum*, cria contas de teste com saldo e registra blocos, transações e eventos. Essa simulação permite experimentar contratos inteligentes, observar custos de gás, verificar *logs* e repetir cenários sem custo financeiro e sem depender de internet pública.

No projeto, ele funcionou como ambiente padrão para compilar, migrar e exercitar os contratos. O endereço *RPC* apontou para a máquina local na porta 7545 e o *network id* usado foi 1337. As implantações de Autoridade e Currículo ocorreram nessa rede e os cenários de emissão, revogação e verificação foram executados e auditados pelas telas do *Ganache*. Quando necessário, o estado foi reiniciado, o que agilizou a correção de erros e a repetição dos testes até alcançar o comportamento esperado.

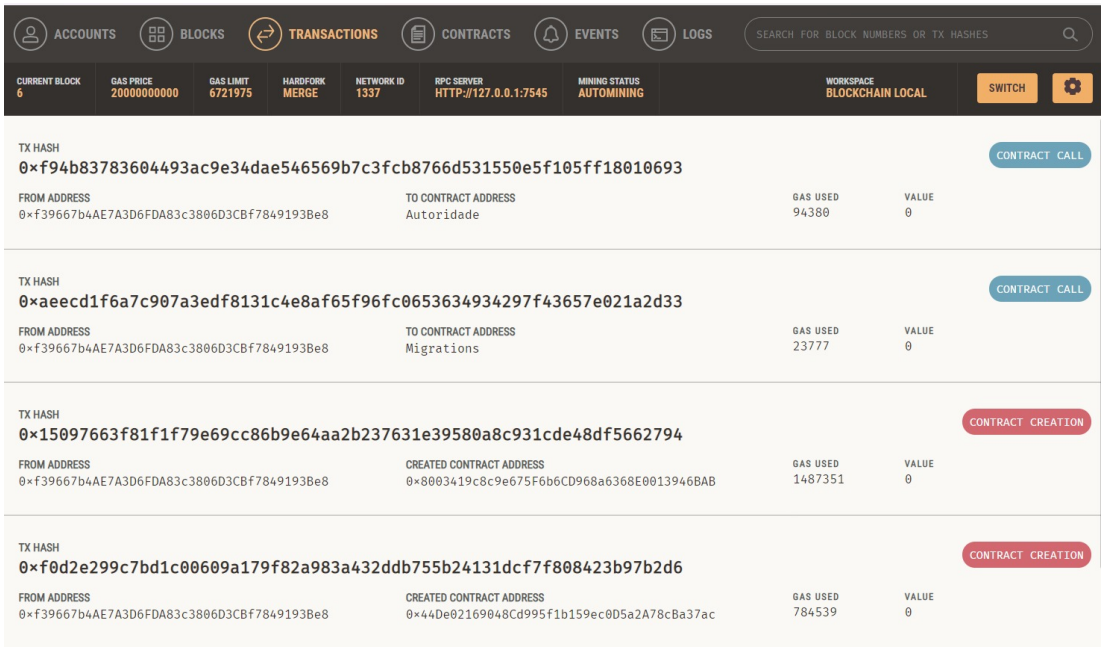
A **Figura 8** apresenta a tela do *Ganache* que confirma a rede local ativa e pronta para testes. No topo aparecem o bloco atual, o preço do gás, o limite de gás e o estado de mineração automática. Ao centro está o endereço do servidor *RPC* em 127.0.0.1:7545 e o *network id* igual a 1337, que são os mesmos usados na configuração do *Truffle* e na carteira. A seção de contas exibe o *mnemônico* gerado para o *workspace* e lista carteiras de teste com seus endereços, saldos e contagem de transações. Essa visão sustenta o uso das contas como emissores de transações, mostra que há saldo suficiente para pagar gás e registra que a rede está produzindo blocos enquanto o sistema é exercitado.



Fonte: imagem do autor.

Figura 8: tela inicial do Ganache.

Na sequência a **Figura 9** mostra a aba de transações do *Ganache*. Ela lista cada operação enviada para a rede local com o identificador da transação, o endereço de origem, o endereço do contrato de destino ou o endereço do contrato criado, o gás consumido e o valor transferido. As duas linhas inferiores' exibem criações de contrato e confirmam a implantação bem-sucedida, pois trazem o rótulo de criação de contrato e mostram os endereços gerados. As duas linhas superiores exibem chamadas a contratos já implantados, uma para Autoridade e outra para *Migrations*, com o rótulo de chamada de contrato e o gás utilizado. Este registro comprova o ciclo completo, implantação dos contratos e chamadas posteriores durante os testes.



Fonte: imagem do autor.

Figura 9: tela de transações do Ganache.

A **Figura 10** mostra a aba de contratos do *Ganache* e confirma que **Autoridade** e **Currículo** foram implantados com endereços visíveis e que *Migrations* executou a etapa inicial, enquanto Diploma não aparece implantado porque é criado sob demanda e **Estruturas**, **IAutoridade** e **ICurriculo** surgem como não implantados por não serem contratos executáveis.

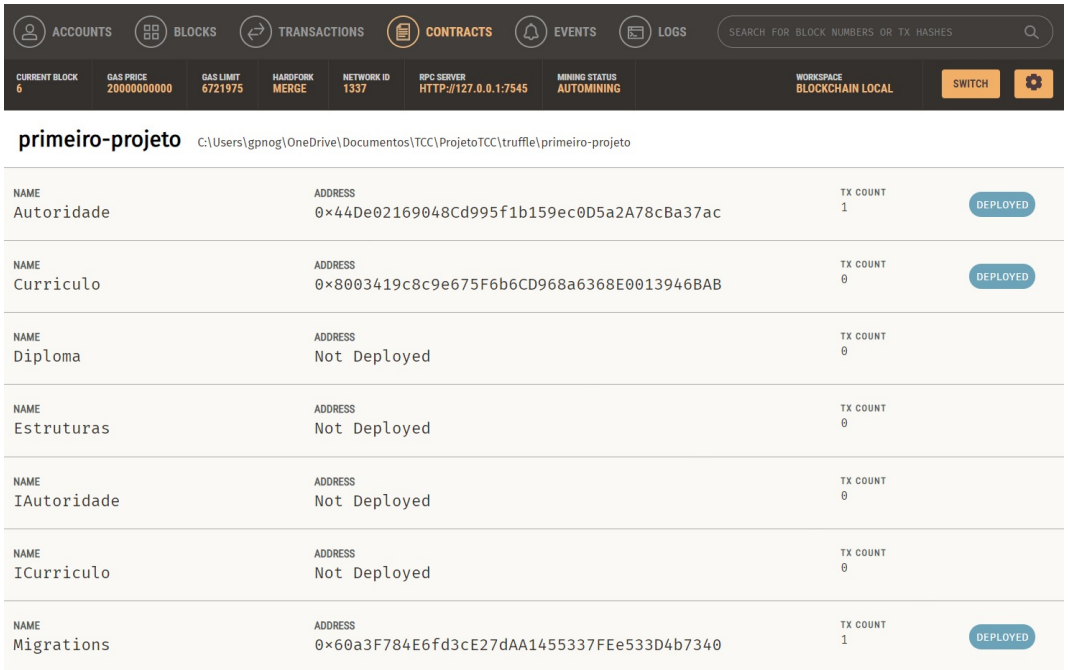


Figura 10: tela de contratos do Ganache.

Estruturas é uma biblioteca com tipos e rotinas de apoio que padronizam o armazenamento e evitam repetição. **IAutoridade** é a *interface* que descreve as funções públicas do contrato de **Autoridade** e garante integração segura. **ICurriculo** é a *interface* que expõe consultas sobre vínculo do aluno e progresso acadêmico para viabilizar a emissão do diploma com baixo acoplamento.

3.5 Web3.js

Web3.js (ETHEREUM FOUNDATION, 2025) é a ponte entre a interface do navegador e a *blockchain* local compatível com a máquina virtual de *Ethereum*. Ele permite que a aplicação leia o estado dos contratos e envie transações assinadas pela carteira do usuário. Com ele, a página inicializa a conexão pelo *provider* da carteira, descobre a conta ativa, cria as instâncias dos contratos a partir do arquivo de interface binária e do endereço implantado e então chama métodos de leitura com *call* e de escrita com *send*. Também fornece *receipts* de transação, eventos e erros para que a interface mostre mensagens de sucesso ou de falha em tempo real. No projeto, ele viabiliza toda a comunicação entre *React* e os contratos Autoridade, Currículo e Diploma, incluindo emissão e revogação no fluxo de autoridade, atualização do progresso no currículo e verificação pública do diploma.

A **Listagem 3.5.1** mostra a inicialização do *Web3* no *front end*. Ao montar o *app*, o *hook useEffect* verifica se existe *window.ethereum*. Se não existir, mostra a mensagem sobre *MetaMask* ausente e encerra. Se existir e ainda não houver instância local, cria *new Web3* com o *provider* da extensão, grava no estado com *setWeb3* e também em *window.web3* para depuração no *console*. Em seguida tenta ler as contas já autorizadas por meio de *ethereum.request* com o método *eth-accounts*, o que não abre janela e apenas retorna contas que já foram expostas ao site. Se vier alguma conta, define a conta ativa e atualiza a lista de contas. Em resumo, esse bloco prepara a aplicação para falar com a *blockchain* assim que a página carrega, sem exigir aprovação imediata do usuário.

Listagem 3.5.1: inicialização do *web3*.

```
useEffect(() => {
   const init = async () => {
     if (!window.ethereum) {
       setMensagem('MetaMask não encontrado. Instale a extensão para
           continuar.');
       return;
     }
     if (!web3) {
       const web3Instance = new Web3(window.ethereum);
       setWeb3(web3Instance);
       window.web3 = web3Instance;
     }
     try {
       const accs = await window.ethereum.request({ method: '
           eth_accounts' });
       if (accs && accs.length) setAccount(accs[0]);
       setAccounts(accs || []);
     } catch (e) { console.warn('Falha ao ler eth_accounts:', e); }
   };
   init();
 }, []);
```

Essa **Listagem 3.5.2** mostra a seleção do *provider* e a conexão inicial. Primeiro tenta usar o objeto *window.ethereum* da *MetaMask* e cria a instância de *Web3*. Em seguida solicita a conexão chamando o método *eth-requestAccounts* para o usuário liberar uma conta ao site. Se o usuário recusar a conexão, retorna um erro amigável. Caso não exista *window.ethereum* mas exista *window.web3*, usa o *currentProvider* herdado. Se nenhum *provider* estiver disponível, aplica um *fallback* para o nó local do *Ganache* no endereço *HTTP* 127.0.0.1, porta 7545. Depois disso busca as contas com *web3.eth.getAccounts*, pega a primeira como conta ativa e interrompe com erro se nenhuma conta for encontrada, pedindo que o usuário verifique a *MetaMask* ou o *Ganache*. Por fim lê o identificador da rede com *web3.eth.net.getId* e registra no *console* para depuração.

Listagem 3.5.2: carregar contratos e rede do *web3*.

```
// Função para carregar Web3 e os contratos
const getBlockchainData = async () => {
  try {
    // Inicializa o Web3 (MetaMask primeiro, fallback para Ganache)
    let web3;

    if (window.ethereum) {
      web3 = new Web3(window.ethereum);
      try {
        // Solicita conexão da MetaMask
        await window.ethereum.request({ method: 'eth_requestAccounts' })
            ;
      } catch (error) {
        console.error("Usuário rejeitou conexão:", error);
        return { error: "Conexão MetaMask rejeitada" };
      }
    } else if (window.web3) {
      web3 = new Web3(window.web3.currentProvider);
    } else {
      // Fallback para Ganache local
      web3 = new Web3('http://127.0.0.1:7545');
    }

    // Pega as contas conectadas
    const accounts = await web3.eth.getAccounts();
    const account = accounts[0];

    if (!account) {
      throw new Error('Nenhuma conta encontrada. Verifique MetaMask/
          Ganache.');
    }

    // Pega o ID da rede
    const networkId = await web3.eth.net.getId();
    console.log("Network ID:", networkId);
```

A **Listagem 3.5.3** cria a instância do contrato Autoridade no lado do navegador usando a *ABI* do artefato compilado e o endereço já implantado na rede atual. A partir desse objeto o aplicativo consegue chamar métodos de leitura com *call* e enviar transações de escrita com *send* usando a conta conectada.

Listagem 3.5.3: criar instância do contrato.

```
const autoridade = new web3.eth.Contract(
     AutoridadeContract.abi,
     deployedNetworkAutoridade.address
   );
```

A **Listagem 3.5.4** realiza o acompanhamento da transação até a confirmação. A função cria uma *promise* e inicia um ciclo de verificação chamado *poll*. Em cada ciclo, consulta *web3.eth.getTransactionReceipt* usando o *hash*. Se o recibo ainda não existir, agenda nova checagem após 800 milissegundos e continua aguardando. Quando o recibo chega, a *promise* é resolvida com o valor booleano de *r.status*, que indica sucesso na cadeia quando verdadeiro. Assim, o *app* consegue exibir mensagens de andamento entre o envio e a confirmação em bloco.

Listagem 3.5.4: envio e acompanhamento da transação.

```
const waitReceipt = async (web3, txHash) => {
  return new Promise((resolve) => {
    const poll = async () => {
      const r = await web3.eth.getTransactionReceipt(txHash);
      if (!r) return setTimeout(poll, 800);
      resolve(Boolean(r.status));
    };
    poll();
  });
};
```

Essa **Listagem 3.5.5** mostra o *deploy* de um novo contrato de Diploma direto no *front end*. Primeiro cria o objeto *deployment* com a *ABI* do Diploma e o *bytecode* e informa os argumentos do construtor na ordem aluno, emissor, currículo e autoridade. Em seguida envia a transação a partir da conta conectada, com limite de gás definido, e aguarda a criação do contrato. Quando a rede retorna a instância, o *app* guarda o objeto do contrato no estado, registra o endereço implantado e atualiza o campo usado para

checagem. Por fim, exibe a mensagem de sucesso com o endereço do novo Diploma ou mostra o erro formatado caso a transação falhe.

Listagem 3.5.5: *deploy* de um contrato direto do *front end*.

```
const deployment = new web3.eth.Contract(DiplomaContract.abi).deploy({
       data: DiplomaContract.bytecode,
       arguments: [addrAlunoParaDiploma, account, curriculo.options.
           address, autoridade.options.address],
     });
     const instance = await deployment.send({ from: account, gas:
         3000000 });
     setDiplomaInst(instance);
     setNovoDiplomaAddr(instance.options.address);
     setAddrContratoDiplomaChecar(instance.options.address);
     setMensagem(" Diploma criado: " + instance.options.address);
   } catch (e) { setMensagem(" X " + revertReason(e)); console.error(e)
       ; }
 };
```

Essa **Listagem 3.5.6** valida a entrada antes de emitir o certificado e padroniza o identificador em *bytes32*. Primeiro usa *web3.utils.isAddress* para garantir que o endereço da instituição é válido. Em seguida, a função *toBytes32* recebe um valor qualquer, converte para *string* e remove espaços. Se vier vazio, lança erro pedindo um *hash* ou um texto. Se for um *hash* no formato correto, com *0x* seguido de 64 caracteres hexadecimais, retorna o mesmo em minúsculas. Se começar com *0x* mas estiver no formato errado, lança erro explicando que precisa ter *0x* mais 64 hexadecimais. Caso contrário, calcula *keccak256* do texto fornecido e devolve o *bytes32* resultante. Assim, o *app* aceita tanto um identificador já em *bytes32* quanto um texto comum, gerando o *hash* de forma segura antes de enviar para o contrato.

Listagem 3.5.6: validação de uma conta para emitir um certificado.

```
if (!web3.utils.isAddress(instituicao)) return setMensagem("X Endereço
    de instituição inválido.");

     const toBytes32 = (v) => {
       const s = String(v ?? "").trim();
       if (!s) throw new Error("Informe um hash ou um texto para gerar
```

```
        o id.");
    if (/^0x[0-9a-fA-F]{64}$/.test(s)) return s.toLowerCase();
    if (s.startsWith("0x")) throw new Error("Hash inválido: precisa
        ser 0x + 64 hex (bytes32).");
    return web3.utils.keccak256(s);
  };
```

3.6 MetaMask

MetaMask (METAMASK, 2025) é a carteira que fornece o provedor *web* para o *Descentralized Application* (DApp) e faz a ponte entre o navegador e a *blockchain*. Ela injeta o objeto *window.ethereum*, guarda as chaves em segurança dentro da extensão e mostra ao usuário cada solicitação de conexão e de assinatura. O *DApp* pede acesso às contas e a *MetaMask* expõe o endereço público somente após o usuário aprovar. Leituras de estado usam chamadas de leitura e não exigem confirmação. Escritas exigem confirmação e assinatura, e a transação é enviada pela própria carteira. A *MetaMask* também controla a rede ativa, como *Ganache* ou outra, e emite eventos de troca de conta e de rede que o *app* observa para atualizar a interface.

Esse bloco da **Listagem 3.6.1** tem duas ações ligadas à *MetaMask*. A primeira é *connectMetaMask*. Ela verifica se existe o objeto *ethereum* no navegador. Se não existir, mostra alerta de *MetaMask* não encontrada e encerra. Se existir, liga o estado de conectando. Pede ao provedor a lista de contas usando o método *eth-requestAccounts*. Ao receber as contas, define a conta ativa como a primeira e guarda a lista completa. Se ainda não havia instância do *Web3*, cria uma nova usando o *provider* da extensão e salva tanto no estado quanto em *window.web3* para depuração. Por fim, envia uma mensagem de sucesso. Se o usuário recusar ou ocorrer erro, registra no *console* e mostra mensagem de falha. Em todos os casos, desliga o estado de conectando. Todo esse fluxo é implementado em *JavaScript*, normalmente com uma função assíncrona em *React* que usa *async/await* e a biblioteca *Web3.js* para comunicar-se com o provedor e deixar *window.web3* disponível para depuração.

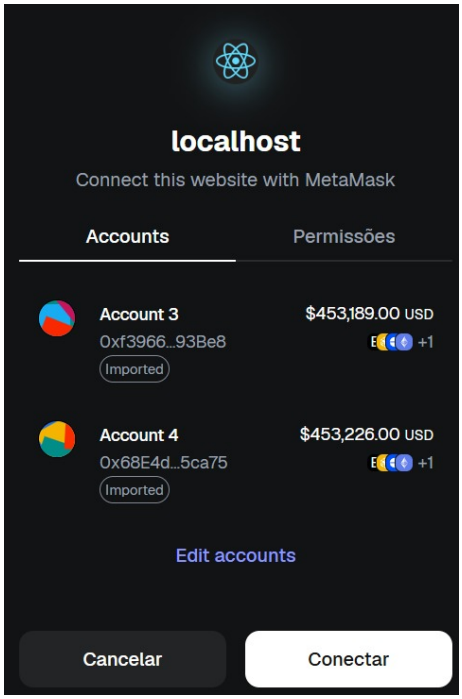
Listagem 3.6.1: verificação de contas e *Ethereum* no *MetaMask*.

```
const connectMetaMask = useCallback(async () => {
   if (!window.ethereum) { alert('MetaMask não encontrada!'); return; }
   try {
     setConnecting(true);
     const accs = await window.ethereum.request({ method: '
        eth_requestAccounts' });
     setAccount(accs?.[0] || null);
     setAccounts(accs || []);
     if (!web3) {
       const web3Instance = new Web3(window.ethereum);
       setWeb3(web3Instance);
       window.web3 = web3Instance;
     }
     setMensagem('Carteira conectada com sucesso.');
   } catch (error) {
     console.error('Erro ao conectar a MetaMask:', error);
     setMensagem('Conexão cancelada ou falhou.');
   } finally { setConnecting(false); }
 }, [web3]);

 const switchAccount = useCallback(async () => {
   if (!window.ethereum) { alert('MetaMask não encontrada!'); return; }
   setConnecting(true);
   try {
     const accs = await window.ethereum.request({ method: 'eth_accounts
        ' });
     setAccounts(accs || []);
     setSelectedAccount('');
     setShowPicker(true);
     setMensagem('Selecione abaixo a conta exposta que o app deve usar.
         Para adicionar outra, use Permissões no MetaMask.');
   } catch (e) {
     console.warn('Erro ao obter contas:', e);
     setMensagem('Não foi possível ler as contas expostas.');
   } finally { setConnecting(false); }
 }, []);
```

A segunda ação é *switchAccount*. Ela também verifica a presença de *ethereum* e ativa o estado de conectando. Em vez de pedir novas permissões, apenas consulta as contas já expostas ao site com o método *eth-accounts*. Atualiza a lista de contas no estado. Limpa a seleção atual. Abre a janela interna do aplicativo para o usuário escolher qual conta quer usar. Mostra uma mensagem explicando que novas contas devem ser liberadas nas permissões da *MetaMask*. Se houver erro, registra no *console* e exibe mensagem apropriada. No fim, desliga o estado de conectando.

Essa janela (ver **Figura 11**) é o pedido de conexão da *MetaMask* para o seu site *localhost*. Ela aparece quando o *app* chama o método *eth-requestAccounts*. Na aba *Accounts*, a carteira lista as contas disponíveis com endereço abreviado e saldo estimado. O usuário escolhe quais contas deseja expor ao site. O botão *Edit accounts* permite ajustar a seleção. Em *Cancelar*, o *app* não recebe nenhuma conta. Em *Conectar*, a *MetaMask* autoriza o site a ver o endereço público das contas marcadas e a usar essas contas ao assinar transações. A aba *Permissions* mostra o que o site poderá acessar e serve para revisar a autorização antes de aceitar.



Fonte: imagem do autor.

Figura 11: permissão feita pelo Metamask.

3.7 React

No projeto, o *React* (META OPEN SOURCE, 2025) é usado com *JavaScript* e *JSX*, que é a sintaxe de marcação dentro do próprio código. A interface é composta por componentes funcionais e todo o comportamento nasce do estado e dos efeitos. O estado guarda a conta conectada, mensagens, contratos carregados, dados de formulário e permissões. Os *hooks* controlam o ciclo de vida e a performance. Um conjunto simples de componentes visuais, como botão, selo, seção e aviso, garante padrão de estilos e de *feedback*. O *app* conversa com a *blockchain* a partir dos efeitos que escutam mudanças de conta e de rede e dispara leituras e escritas. A página de diploma orquestra um fluxo por etapas, valida o emissor *on-chain*, compara com a conta ativa e só então libera a geração da imagem do diploma em *canvas* com pré-visualização e *download* imediato. Tudo reage em tempo real graças à ligação direta entre estado e renderização.

É utilizado também o *React Router*, que organiza a experiência em páginas lógicas e permite navegar entre autoridade, currículo, diploma e verificação, mantendo o mesmo estado global e um cabeçalho único para toda a aplicação.

3.8 React Router

React Router (REMIX SOFTWARE, 2025) cuida da navegação em uma *Single Page Application*[1], trocando o conteúdo exibido sem recarregar a página. Ele observa a *URL*, decide qual componente renderizar e mantém a interface fluida. Com isso, é possível ter um cabeçalho fixo, um estado global preservado e páginas lógicas como **Autoridade**, **Currículo**, **Diploma** e **Verificar**. A navegação é declarativa, os *links* usam *NavLink* para aplicar estilo ativo e a mudança de rota dispara apenas a atualização do componente-alvo.

[1]Uma Single Page Application (SPA) é uma aplicação web que carrega uma única página inicial e atualiza apenas partes do conteúdo conforme o usuário interage, sem recarregar a página inteira. Isso torna a navegação mais rápida e fluida.

A **Listagem 3.8.1** mostra o bloco que define as páginas dentro do *React Router*. A primeira linha usa *Navigate* para que a rota raiz / redirecione para */autoridade*. Assim, sempre que o usuário acessa o endereço base, cai direto na página da Autoridade.

Listagem 3.8.1: definição de rota para **Autoridade**.

```
\\PÁGINAS
       <Routes>
         <Route path="/" element={<Navigate to="/autoridade" replace />}
             />
         <Route
           path="/autoridade"
           element={
             <AutoridadePage
               autoridade={autoridade}
               autorizada={autorizada}
               instituicao={instituicao} setInstituicao={setInstituicao}
               hashCertificado={hashCertificado} setHashCertificado={
                   setHashCertificado}
               validade={validade} setValidade={setValidade}
               instituicaoRevogar={instituicaoRevogar}
                   setInstituicaoRevogar={setInstituicaoRevogar}
               instituicaoVerificar={instituicaoVerificar}
                   setInstituicaoVerificar={setInstituicaoVerificar}
               verificacaoResultado={verificacaoResultado}
               emitirCertificado={emitirCertificado}
               revogarCertificado={revogarCertificado}
               verificarCertificado={verificarCertificado}
               listarInstituicoesRevogadas={listarInstituicoesRevogadas}
               revogados={revogados}
               revogadosCarregado={revogadosCarregado}
             />
           }
         />
```

Em seguida, a rota *path="/autoridade"* renderiza o componente *AutoridadePage* e recebe um pacote de propriedades vindo do *App*. Essas propriedades carregam referências aos contratos e aos estados, como *autoridade* e *autorizada*, os valores e *setters* dos

campos de formulário, como *instituicao*, *hashCertificado* e *validade*, além das funções de ação, como *emitirCertificado*, *revogarCertificado*, *verificarCertificado* e *listarInstituicoesRevogadas*. Também passam os dados de resultado, como *verificacaoResultado* e a lista *revogados* com o sinalizador *revogadosCarregado*. Na prática, o *App* centraliza o estado e injeta tudo que a página precisa para ler entradas do usuário, chamar o contrato e exibir o retorno.

3.9 Node.js e npm

Node.js (OPENJS FOUNDATION, 2025) é o ambiente que executa *JavaScript* fora do navegador. Ele roda o servidor de desenvolvimento do *React*, compila o código para produção, executa scripts na linha de comando e permite usar bibliotecas como *Web3*, *Truffle* e *Ganache*. Com *Node.js* conseguimos compilar *JSX* e *CSS*, empacotar os arquivos e automatizar tarefas como *lint*, teste e *build*.

Já o *npm* é o gerenciador de pacotes do *Node.js*. Ele lê o arquivo *package.json*, instala e atualiza dependências, registra versões no *package-lock.json* e disponibiliza scripts chamados com *npm run* para iniciar o *app*, gerar o *build*, rodar testes e outras rotinas do projeto. Isso garante um fluxo de trabalho consistente e reproduzível em qualquer máquina com *Node* instalado.

3.10 Geração de imagem do diploma

A geração da imagem do diploma foi implementada para criar um documento visual padronizado, pronto para compartilhamento, sem necessidade de servidores. A composição ocorre diretamente no navegador, utilizando o elemento *canvas* do *HTML5*, permitindo exportação para *PNG*. Essa abordagem reduz custos operacionais [2], preserva a privacidade e possibilita que a instituição emita e baixe o arquivo imediatamente. Além disso, mantém a identidade visual do projeto com molduras, tipografia e *layout* consistentes, permitindo pequenas personalizações, como nome do aluno, curso, instituição emissora e data.

[2]Custo operacional, nesse contexto, significa as despesas associadas a manter servidores ou infraestrutura para gerar e armazenar diplomas. Como a imagem é produzida diretamente no navegador, não há gasto com servidores, reduzindo esse custo a praticamente zero.

O processo desenha os elementos do diploma diretamente no *canvas* e grava o resultado como endereço de dados em *PNG* para pré-visualização e *download*. Os campos exibidos na tela alimentam o desenho em tempo real, o que simplifica a conferência antes de salvar. O conteúdo permanece no dispositivo do usuário, sem trânsito por serviços externos, o que atende requisitos de simplicidade e segurança.

A geração do diploma é controlada por permissões: o sistema verifica na blockchain o endereço do emissor do contrato e só permite criar a imagem quando a conta conectada corresponde a esse emissor. O diploma pode incluir um código *QR* que direciona à página pública de verificação, permitindo que terceiros confirmem sua validade. O *QR code* é atualmente gerado por uma *API* externa, que retorna a imagem do código, a qual é desenhada diretamente no *canvas* e integrada à imagem do diploma. Todo o processo mantém a identidade visual do projeto, incluindo molduras, tipografia, *layout* consistente e personalizações como nome do aluno, curso, instituição emissora e data.

A **Listagem 3.10.1** mostra o código que descobre o endereço do emissor direto no contrato do diploma. Primeiro a função cria uma instância do contrato usando a *ABI* e o endereço informado. Em seguida tenta uma lista de nomes comuns de *getters*, como *concedente*, *emissor*, *issuer*, *owner*, *instituicao* e *getOwner*. Para cada nome verifica se o método existe, chama a leitura e, se o valor retornado for um endereço válido, devolve esse valor imediatamente.

Listagem 3.10.1: verificação do endereço do emissor do diploma.

```
sync function getIssuerFromDiploma(web3, contractAddr) {
  const inst = new web3.eth.Contract(DiplomaContract.abi, contractAddr);
  //getters comuns
  const prefer = ['concedente', 'emissor', 'issuer', 'owner', '
     instituicao', 'getOwner'];
  for (const name of prefer) {
    if (inst.methods[name]) {
      try {
        const v = await inst.methods[name]().call();
        if (web3.utils.isAddress(v)) return v;
      }
      catch {}
    }
  }
```

```
14    const abi = inst.options?.jsonInterface || [];
15    const cand = abi.find(f =>
16      f.type === 'function' &&
17      (f.stateMutability === 'view' || f.constant === true) &&
18      (f.inputs || []).length === 0 &&
19      (f.outputs || []).length === 1 &&
20      f.outputs[0].type === 'address' &&
21      /(concedente|emissor|issuer|owner|instituicao)/i.test(f.name)
22    );
23    if (cand) {
24      try {
25        const v = await inst.methods[cand.name]().call();
26        if (web3.utils.isAddress(v)) return v;
27      }
28      catch {}
29    }
30
31    throw new Error('Não foi possível identificar o emissor no ABI do
          Diploma.');
32  }
```

Na sequência (ver **Listagem 3.10.2**) é apresentada a função *gerarImagemDiplomaPNG* que compõe, no cliente, um diploma em formato de imagem usando o canvas do *HTML5* com dimensões de 1654×1170 pixels. Primeiro cria-se o elemento *canvas* e obtém-se o contexto 2D armazenadado na variável *ctx*; em seguida aplica-se um gradiente de fundo e desenham-se duas molduras arredondadas (função *drawRoundedRect*) para gerar a borda visual. Uma faixa superior sólida é pintada e um título central é renderizado com fonte serifada em negrito; depois posiciona-se o nome da instituição, uma linha separadora e um emblema (função *drawMedal*). O bloco de texto principal é inserido com quebra manual por *wrapTextCanvas* para respeitar a largura disponível; nomes e subtítulos são desenhados com alinhamentos e estilos distintos. Campos técnicos (emissor, contrato, data) usam uma função auxiliar *drawLabelValue* que mede a largura do rótulo (*measureText*) e quebra o valor em linhas com *breakLines*, garantindo alinhamento e limites máximos de largura. A seção de assinaturas é desenhada com linhas e rótulos posicionados em colunas calculadas a partir da largura do *canvas*. Caso a inclusão do *QR* esteja habilitada, a função monta uma URL para uma *API* externa

(*api.qrserver.com*), carrega a imagem gerada (*loadImage*) e a desenha no canvas dentro de um quadro arredondado, adicionando um rótulo “Verificação”; todo esse passo está protegido por *try/catch* para evitar falhas no desenho principal se a requisição falhar. Em suma, o trecho organiza a composição gráfica com cores, tipografia, *layout* responsivo ao conteúdo textual e elementos vetoriais, assim preparando a imagem final do diploma inteiramente no navegador.

Listagem 3.10.2: geração da imagem do diploma usando *canvas*.

```
1
2  async function gerarImagemDiplomaPNG({
3    alunoNome,
4    cursoNome,
5    instituicaoNome,
6    emissorAddress,
7    contratoAddress,
8    dataEmissaoStr,
9    urlVerificacao,
10   incluirQR = true
11 }) {
12
13   const W = 1654, H = 1170;
14   const canvas = document.createElement('canvas');
15   canvas.width = W; canvas.height = H;
16   const ctx = canvas.getContext('2d');
17
18   const grad = ctx.createLinearGradient(0, 0, W, H);
19   grad.addColorStop(0, '#ffffff');
20   grad.addColorStop(1, '#f7faff');
21   ctx.fillStyle = grad;
22   ctx.fillRect(0, 0, W, H);
23
24   drawRoundedRect(ctx, 20, 20, W - 40, H - 40, 24, '#dfe9ff', 6);
25   drawRoundedRect(ctx, 40, 40, W - 80, H - 80, 18, '#b9ccff', 2);
26
27   const faixaTop = 80, faixaH = 100;
28   ctx.fillStyle = '#0b3d91';
29   ctx.fillRect(60, faixaTop, W - 120, faixaH);
```

```
30
31  ctx.fillStyle = '#ffffff';
32  ctx.font = "bold 64px 'Times New Roman', serif";
33  ctx.textAlign = 'center';
34  ctx.fillText('DIPLOMA ACADÊMICO', W / 2, faixaTop + 70);
35
36  const instY = faixaTop + faixaH + 80;
37
38  ctx.fillStyle = '#0b3d91';
39  ctx.textAlign = 'left';
40  ctx.font = "bold 42px 'Times New Roman', serif";
41  ctx.fillText(instituicaoNome || 'Instituição de Ensino', 240, instY);
42
43  ctx.strokeStyle = '#b9ccff';
44  ctx.lineWidth = 2;
45  drawLine(ctx, 240, instY + 14, W - 100, instY + 14);
46
47  drawMedal(ctx, 150, faixaTop + faixaH + 60, 55, '#0b3d91');
48
49  const blocoX = 120;
50  let y = instY + 70;
51
52  ctx.fillStyle = '#223a63';
53  ctx.font = '22px Georgia, serif';
54  ctx.textAlign = 'left';
55  wrapTextCanvas(
56    ctx,
57    `Certifica-se que ${alunoNome || 'NOME DO ALUNO'}, concluiu com êxito o curso de ${cursoNome || 'NOME DO CURSO'}, atendendo aos critérios acadêmicos estabelecidos pela instituição e estando apto(a) àcolação de grau.`,
58    blocoX,
59    y,
60    W - 120 * 2,
61    34
62  );
63
```

```
64  y += 170;
65  ctx.textAlign = 'center';
66  ctx.fillStyle = '#0b3d91';
67  ctx.font = 'bold 54px Georgia, serif';
68  ctx.fillText(alunoNome || 'NOME DO ALUNO', W / 2, y);
69
70  y += 56;
71  ctx.textAlign = 'center';
72  ctx.fillStyle = '#223a63';
73  ctx.font = 'italic 28px Georgia, serif';
74  ctx.fillText(cursoNome ? `Curso: ${cursoNome}` : 'Curso: _', W / 2, y)
        ;
75
76  y += 100;
77  ctx.textAlign = 'left';
78  const GAP = 16;
79  const LEFT = blocoX;
80  const MONO = "ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, '
        Liberation Mono', 'Courier New', monospace";
81
82  y = drawLabelValue(ctx, {
83    label: 'Emissor:',
84    value: emissorAddress || '_',
85    y,
86    left: LEFT,
87    labelFont: 'bold 22px Georgia, serif',
88    valueFont: `22px ${MONO}`,
89    gap: GAP,
90    maxWidth: W - LEFT - 120
91  });
92
93  y = drawLabelValue(ctx, {
94    label: 'Contrato:',
95    value: contratoAddress || '_',
96    y,
97    left: LEFT,
98    labelFont: 'bold 22px Georgia, serif',
```

```
99    valueFont: `24px ${MONO}`, // AQUI: maior
100   gap: GAP,
101   maxWidth: W - LEFT - 120
102 });
103
104 y = drawLabelValue(ctx, {
105   label: 'Data de Emissão:',
106   value: dataEmissaoStr || formatDateBr(new Date()),
107   y,
108   left: LEFT,
109   labelFont: 'bold 22px Georgia, serif',
110   valueFont: '22px Georgia, serif',
111   gap: GAP,
112   maxWidth: 480
113 });
114
115 const linhaY = H - 250;
116 const col1 = W / 2 - 320;
117 const col2 = W / 2 + 80;
118
119 ctx.strokeStyle = '#94A3B8';
120 ctx.lineWidth = 1.6;
121 drawLine(ctx, col1, linhaY, col1 + 300, linhaY);
122 ctx.textAlign = 'center';
123 ctx.fillStyle = '#223a63';
124 ctx.font = '20px Georgia, serif';
125 ctx.fillText('Diretor Acadêmico', col1 + 150, linhaY + 30);
126
127 drawLine(ctx, col2, linhaY, col2 + 300, linhaY);
128 ctx.fillText('Coordenador do Curso', col2 + 150, linhaY + 30);
129
130 if (incluirQR && urlVerificacao) {
131   try {
132     const qrUrl = `https://api.qrserver.com/v1/create-qr-code/?size
          =230x230&margin=0&data=${encodeURIComponent(urlVerificacao)}`;
133     const qrImg = await loadImage(qrUrl);
134     const qrSize = 230;
```

```
135       const qx = W - 100 - qrSize;
136       const qy = H - 100 - qrSize;
137
138       ctx.strokeStyle = '#b9ccff';
139       ctx.lineWidth = 2;
140       drawRoundedRect(ctx, qx - 10, qy - 10, qrSize + 20, qrSize + 20,
              12, '#b9ccff', 2);
141
142       ctx.drawImage(qrImg, qx, qy, qrSize, qrSize);
143
144       ctx.textAlign = 'center';
145       ctx.fillStyle = '#223a63';
146       ctx.font = '18px Georgia, serif';
147       ctx.fillText('Verificação', qx + qrSize / 2, qy - 18);
148       ctx.textAlign = 'left';
149     } catch {
150     }
151   }
```

Capítulo 4

Resultados obtidos

Ao longo do desenvolvimento validamos o comportamento de cada contrato e confirmamos que as regras de negócio foram aplicadas de forma consistente. No contrato **Autoridade** verificamos a emissão de certificação para instituições, a revogação dessa certificação, a checagem do *status* de um endereço e a listagem de endereços revogados. Testes com múltiplas contas mostraram que somente uma instituição certificada pode emitir e revogar, e que endereços não certificados recebem resposta negativa imediata na verificação. Também confirmamos que a troca de conta no *front end* atualiza o *status* sem necessidade de reconectar permissões.

No contrato **Curriculo** confirmamos o registro de estudantes e de suas disciplinas, a marcação de disciplina como concluída e a verificação de aptidão para graduação. Os experimentos demonstraram que somente a instituição que registrou o estudante consegue marcar a conclusão das disciplinas daquele estudante, bloqueando qualquer tentativa de outra instituição. A função de aptidão para graduar considerou corretamente o conjunto de disciplinas concluídas, retornando que o aluno é apto apenas quando todos os requisitos foram atendidos. Essa checagem serviu de base para todos os fluxos dependentes, garantindo que nenhum aluno com pendências chegasse à etapa de diploma.

No contrato **Diploma** testamos a criação ou o carregamento do contrato individual do aluno, a emissão do diploma, a revogação e a verificação de validade. Os testes confirmaram três restrições centrais, o *deployer* precisa ser instituição certificada e ser o próprio emissor, o estudante precisa pertencer àquela instituição, e a emissão só é autorizada quando o **Curriculo** informa que o aluno apto a graduar. As tentativas

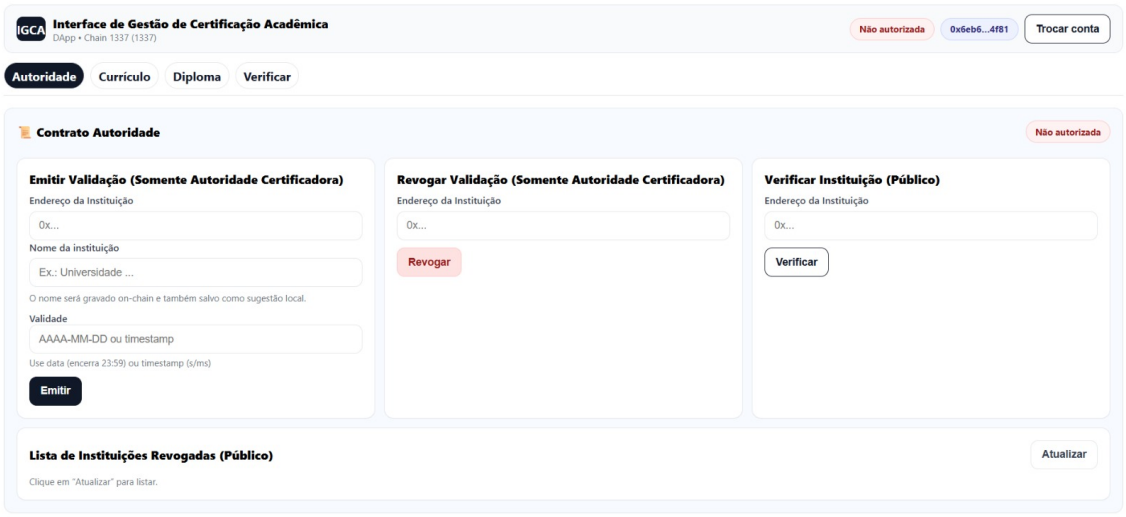
de emitir para alunos de outra instituição foram corretamente barradas, assim como as tentativas de revogar e emitir feitas por contas sem certificação. A consulta de validade refletiu em tempo real os eventos de emissão e revogação.

No *front end* validamos os fluxos de **Autoridade**, **Curriculo** e **Diploma** com Web3 e MetaMask, incluindo a troca dinâmica de contas. A rota pública de verificação respondeu aos testes fornecendo o resultado a partir do endereço informado e o rodapé sincronizou as mensagens de sucesso ou erro conforme o caso. A seção de listagem de instituições revogadas exibiu os endereços um por linha e apresentou o total agregado, o que facilitou auditoria e conferência.

Implementamos e exercitamos a geração da imagem do diploma com código de verificação por QR vinculado à rota pública. Os testes confirmaram a restrição adicional, somente a instituição que emitiu o diploma consegue gerar a imagem daquele diploma. A imagem refletiu os dados de aluno, curso, instituição e data, e o código direcionou para a verificação do diploma no aplicativo. Em tentativas com contas não emissoras a geração foi recusada, mantendo a coerência com as regras de controle de origem.

Por fim, testamos cenários de borda para garantir robustez. A emissão falhou com mensagens claras quando o aluno não estava apto, quando a conta não era certificada ou quando a instituição não correspondia do aluno. A marcação de disciplina concluiu apenas quando chamada pela instituição correta, e a verificação de certificados e diplomas retornou estados consistentes após revogações. Esses resultados demonstram que as funcionalidades principais e suas restrições estão operando de forma integrada, protegendo a autoria institucional e assegurando que somente alunos aptos recebam diplomas válidos.

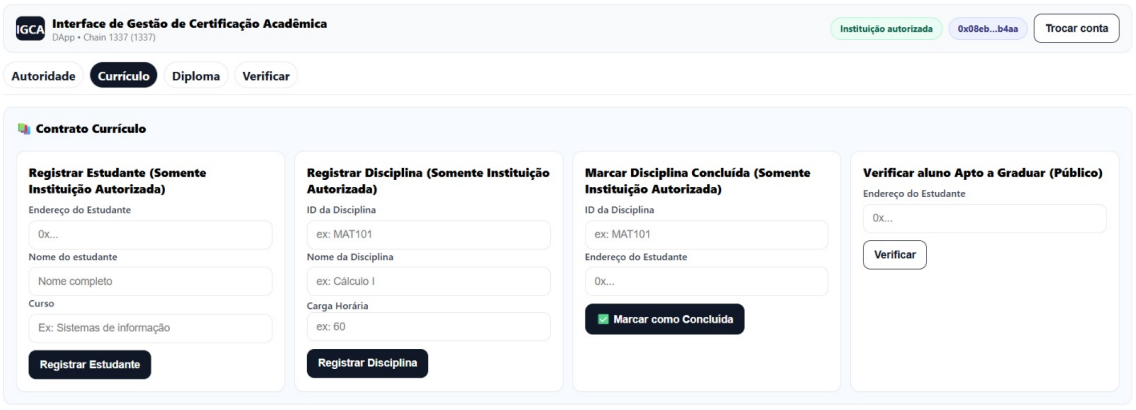
A **Figura 12** mostra a tela principal do contrato de autoridade. Aqui são realizadas as validações feitas pela autoridade certificadoras para as instituições de ensino. Nessa tela são exibidas as funções de *Emitir Validação*, *Revogar Validação* que são exclusivas da autoridade certificadora; e as funções *Verificar Instituição* e *Lista de Instiuições Revogadas* que são funções públicas.



Fonte: imagem do autor.

Figura 12: tela principal do contrato de autoridade.

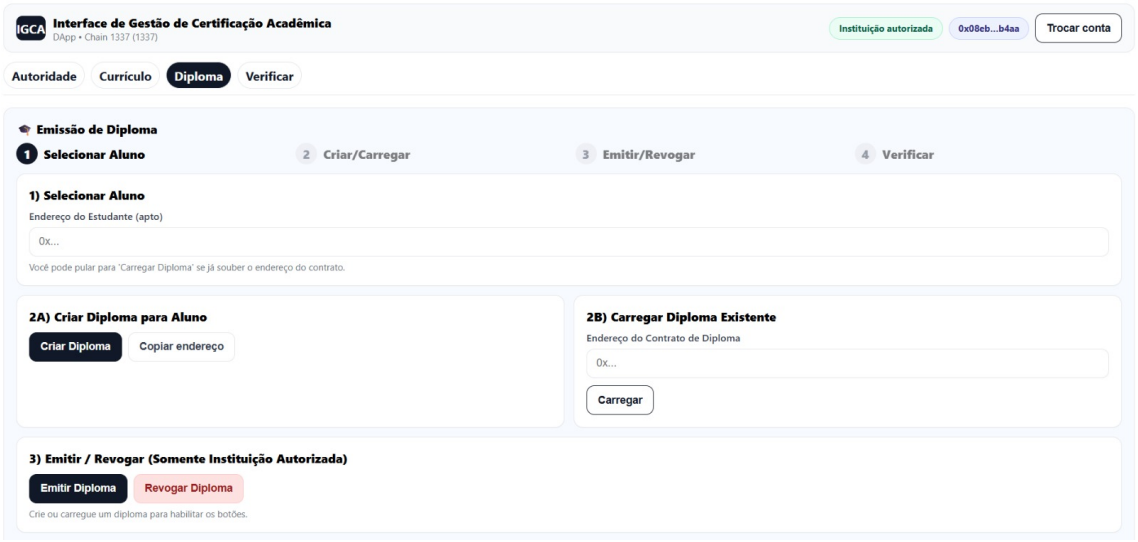
A **Figura 13** apresenta a tela principal do contrato de currículo. Nessa tela é exibida as funções de *Registrar aluno*, *Registrar Disciplina*, *Marcar Disciplina como concluída* que são funções restritas somente para instituições autorizadas; e a função *Verificar aluno Apto a Graduar* que é uma função pública.



Fonte: imagem do autor.

Figura 13: tela principal do contrato de currículo.

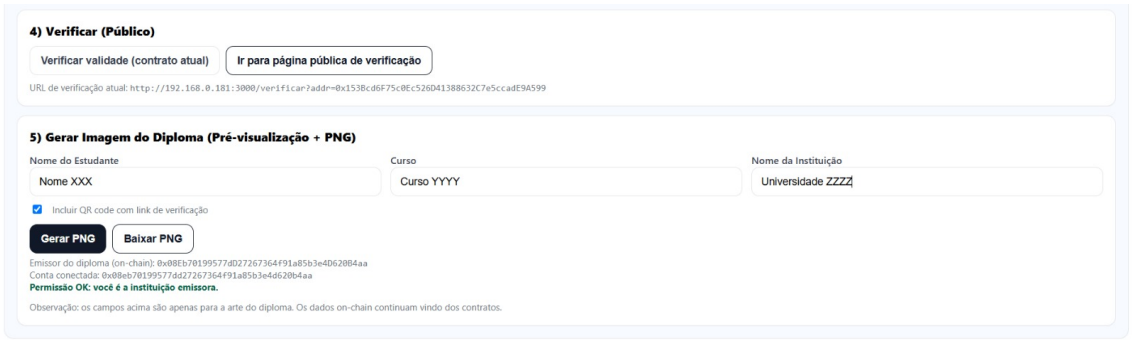
A **Figura 14** exibe a tela inicial do contrato de diploma. Aqui é feito um "passo a passo"para criação de um diploma, desde a seleção do aluno que está apto a graduar até a emissão do diploma, junto com a possível revogação do mesmo.



Fonte : imagem do autor.

Figura 14: tela principal do contrato de diploma.

A **Figura 15** apresenta mais algumas funções que tem na tela do contrato de diploma; É uma continuação direta das outras funções iniciais desse contrato. Nessa imagem é exibido a função para *Verificar o certificado* e por último a função para gerar a imagem do diploma, baseado em todo registro feito anteriormente.



Fonte: imagem do autor.

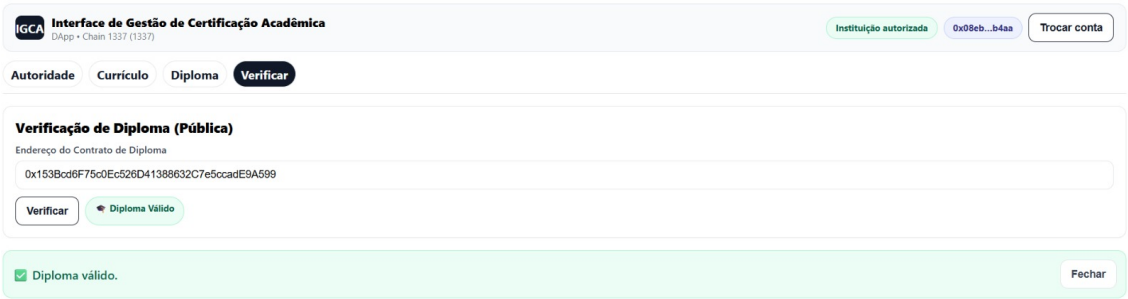
Figura 15: tela principal do contrato de diploma.

A **Figura 16** exibe a imagem do diploma já gerada após a criação e emissão do diploma para o aluno. Na imagem tem informações como o nome do aluno, curso, nome da instituição, data de emissão, o endereço do emissor do diploma (instituição) e endereço do diploma. É possível inserir um código *QR* para a verificação do diploma.

Fonte: imagem do autor.

Figura 16: imagem do diploma gerada.

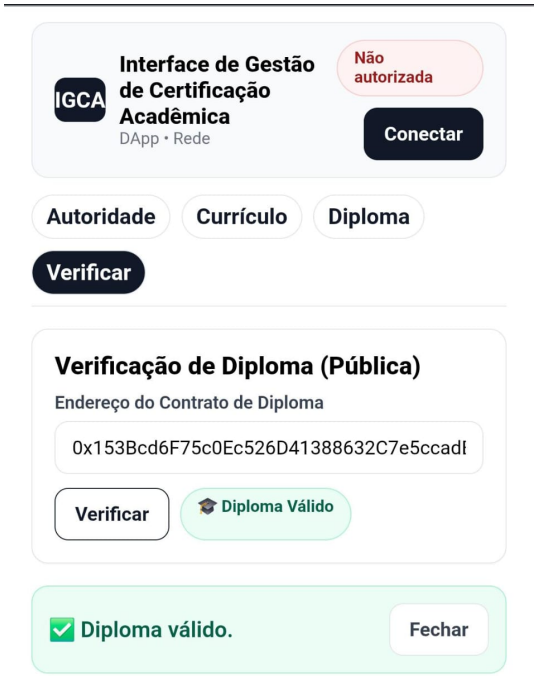
A **Figura 17** apresenta a tela de verificação do diploma. É exibido apenas uma função para a verificação usando o endereço do diploma já emitido.



Fonte: imagem do autor.

Figura 17: tela principal do aba de verificação do diploma.

A **Figura 18** mostra a tela de verificação do diploma visto em um celular após ler o código *QR*.



Fonte: imagem do autor.

Figura 18: tela de verificação após ler o QR code pelo celular.

Capítulo 5

Conclusão

O projeto evidencia que diplomas acadêmicos podem ser emitidos e verificados com uma camada de confiança baseada em *blockchain*, assegurando autenticidade, integridade e verificabilidade independente. A base teórica contemplou criptografia de chave pública, registros imutáveis e o papel de uma autoridade certificadora implementada por contratos inteligentes. No protótipo, essa autoridade é centralizada em uma única conta administrativa; essa escolha não compromete as propriedades de segurança do registro, pois a integridade, a autenticidade e a auditabilidade permanecem garantidas pelas assinaturas e pela imutabilidade da rede, mas concentra a governança e cria um ponto único de decisão. A separação de responsabilidades entre os módulos Autoridade, Currículo e Diploma estabeleceu um nexo claro entre governança institucional e comprovação técnica, permitindo rastreabilidade do início ao fim do processo.

A implementação consolidou regras explícitas de autorização e vínculo entre aluno e instituição. O contrato Autoridade concentra a certificação de quem pode atuar como emissor. O contrato Currículo mantém o histórico acadêmico e o critério de aptidão para graduação. O contrato Diploma realiza a emissão, consulta de validade e eventual revogação, sempre condicionado às informações dos demais módulos. Essa orquestração garante coerência entre política institucional e estado *on chain*, reduzindo ambiguidades e prevenindo emissões indevidas.

No cliente *web*, a aplicação React organiza os fluxos em páginas dedicadas por meio de React Router e integra o provedor de contas com Web3 para assinar e enviar transações. A interface prioriza clareza de status, com mensagens e indicadores que refletem o resultado das interações com a rede. O fluxo de verificação pública foi

exposto em rota específica, permitindo consulta por endereço do diploma e oferecendo transparência para qualquer parte interessada.

A geração da imagem do diploma complementa a prova criptográfica com uma apresentação legível fora da cadeia. O artefato inclui dados essenciais do estudante, curso, instituição e data, além de um código de verificação que direciona o usuário para a rota pública de checagem no sistema. Essa abordagem aproxima a experiência cotidiana da verificação técnica, conectando documentos compartilháveis com a fonte de verdade registrada na rede.

Como trabalhos futuros, prevê-se a ampliação das funcionalidades do sistema, incluindo a integração com bases acadêmicas existentes, a automação de fluxos administrativos e a adoção de padrões internacionais de interoperabilidade para certificações digitais. Também se considera a implementação de módulos avançados de auditoria e visualização de histórico, além de melhorias na experiência do usuário e na escalabilidade dos contratos inteligentes. Quanto à adoção do sistema em uma universidade real, estima-se que a complexidade envolveria a migração de dados, a adaptação aos processos internos e a capacitação das equipes responsáveis. Em termos de desempenho, a execução de registros em larga escala dependeria da infraestrutura blockchain utilizada: redes privadas ou permissionadas permitiriam milhares de transações por segundo, garantindo tempos de processamento adequados mesmo para grandes volumes de emissão e validação de certificados acadêmicos.

REFERÊNCIAS

ABMES, R. **Seminário ABMES - Abril de 2019.** ABMES, 2023. Acessado em 05 de agosto de 2023. Disponível em: <https://www.youtube.com/watch?time_continue=1650&v=YUHXYD6-Bnw>.

ADÃO, J. M. R.; SILVEIRA, S. I. da; SILVEIRA, D. R. da. Uniesp.são paulo. **A função do algoritmo hash md4.**, 2015.

AKIAU, A. M. *et al.* **Desenvolvimento de um smart contract para crowdfunding utilizando a rede de blockchain ethereum**, Blumenau, SC., 2023.

BLOCKCERTS. 2016. MIT Media Lab and Partners. Acessado em 16 de Setembro de 2023. Disponível em: <https://www.blockcerts.org/about.html>.

BRASIL. **LEI No 8.159, DE 8 DE JANEIRO DE 1991**, 1991. Acessado em 22 de Setembro de 2023. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l8159.htm>.

BRASIL. **Universidade Federal da Paraíba passa a emitir diplomas digitais assinados com certificado ICP-Brasil**. GOV BR, 2019. Acessado em 29 de setembro de 2025. Disponível em: <https://abre.ai/govbrufpb>.

BRASIL, M. **PF prende 3 em nova fase de operação contra diplomas falsos de medicina**. G1, 2023. Acessado em 04 de Julho de 2023. Disponível em: <https://g1.globo.com/rj/rio-de-janeiro/noticia/2023/06/20/operacao-catarse-2-da-pf.ghtml>.

BUBENSHCHYKOV, M.; CONTRIBUTORS. **html-to-image: Generate images from DOM nodes**. 2025. Acessado em 14 de outubro de 2025. Disponível em: <https://github.com/bubkoo/html-to-image>.

CAGLIARI, A.; SILVA, G.; SANTOS, L. **Uso da Blockchain para gerenciamento de certificados**, Universidade Presbiteriana Mackenzie, 2022.

COOPER, D. *et al.* **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**. 2008. RFC 5280. Acessado em 14 de outubro de 2023. Disponível em: <https://www.rfc-editor.org/rfc/rfc5280>.

COPALO, E. D. R. Icp-brasil. **Revista CEJ**, v. 7, n. 20, p. 58–66, 2003.

CORMEN, T. *et al.* **Algoritmos-Teoria e Prática**. 3a. edição. [S.l.]: Editora Campus, 2012.

CROSBY, M. *et al.* **Blockchain Technology: Beyond Bitcoin**. [S.l.], 2016. Acessado em 14 de Setembro de 2023. Disponível em: <https://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf>.

DORNELES, S. L.; CORRÊA, R. F. Gestão de documentos digitais em aplicações de certificação digital. **Informação arquivística**, v. 2, n. 2, p. 3–31, 2013.

DUBROWSKY, A. **TRANSFORMAÇÃO DIGITAL NAS INSTITUIÇÕES PRIVADAS DE ENSINO SUPERIOR BRASILEIRAS: Proposta para Autenticação de Diplomas Digitais de Graduação por meio de Blockchain**. 2019.

ETHEREUM FOUNDATION. **Web3.js Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://web3js.readthedocs.io/>.

FIPS, P. 186-2. digital signature standard (dss). **National Institute of Standards and Technology (NIST)**, v. 20, n. 13, p. 5, 2000.

FRIEDRICH, D. M.; MEDINA, R. D. Certificação digital acadêmica: Implantação do sistema de gerenciamento de certificados digitais icpedu na ufsm. **RENOTE**, v. 5, n. 2, 2007.

GRECH, A. F. C. A. **Blockchain in education**, 2017.

GREVE, F. G. *et al.* Blockchain e a revolução do consenso sob demanda. **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)-Minicursos**, 2018.

GUILHEN, S. N. Um servico de autorizacao para um servidor ejb utilizando certificados de atributos x.509. 2006.

GUIMARãES, A. **TRT-RJ identifica 17 certificados digitais suspeitos de fraude; golpe pode ultrapassar os R 4 milhões**. G1, 2023. Acessado em 14 de Setembro de 2023. Disponível em: <https://abre.ai/trtcertificados>.

HAUCK, F. **OpenID for Verifiable Credentials: formal security analysis using the Web Infrastructure Model**. Tese (Doutorado) — University of Stuttgart, 2023.

Instituto Nacional de Tecnologia da Informação (ITI). **Diretrizes e normas da ICP-Brasil (DOC-ICP-02): Requisitos para Certificados Digitais**. 2023. Definições e requisitos para certificados no âmbito da ICP-Brasil. Disponível em: <https://www.gov.br/iti/pt-br>.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information and documentation — Records management — Part 1: Concepts and principles**. 2016. Acessado em 11 de outubro de 2025. Disponível em: <https://www.iso.org/obp/ui/>.

ISO/IEC. **QR Code bar code symbology specification**. 2015.

JURIDOC. **O que são e como funcionam os smart contracts ou contratos inteligentes?** 2022. Acessado em 14 de Setembro de 2023. Disponível em: <https://www.juridoc.com.br/blog/legaltech/12409-o-que-sao-smart-contracts-contratos-inteligentes/>.

KAZIENKO, J. F. *et al.* **Assinatura digital de documentos eletrônicos através da impressao digital**, Florianópolis, SC, 2003.

LAMOUNIER, L. **Quem É Nick Szabo? O Mago Dos Contratos Inteligentes**. 101 Blockchains, 2018. Acessado em 18 de setembro de 2023. Disponível em: <https://101blockchains.com/pt/nick-szabo-contratos-inteligentes/>.

LAURENCE, T. **Blockchain for dummies**. [S.l.]: John Wiley & Sons, 2023.

LIN, I.-C.; LIAO, T.-C. A survey of blockchain security issues and challenges. **Int. J. Netw. Secur.**, v. 19, n. 5, p. 653–659, 2017.

MASCARENHAS, J. Z.; VIEIRA, A. B.; ZIVIANI, A. Análise da rede de transações do ethereum. In: SBC. **Anais do I Workshop em Blockchain: Teoria, Tecnologias e Aplicações**. [S.l.], 2018.

MENEZES, A. J.; OORSCHOT, P. C. V.; VANSTONE, S. A. **Handbook of applied cryptography**. [S.l.]: CRC press, 2018.

META OPEN SOURCE. **React Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://react.dev/>.

METAMASK. **MetaMask Developer Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://docs.metamask.io/>.

MIRANDA, D. S. **Blockchain na educação: uso da tecnologia como prova de existência de diplomas e certificados**, 2019.

MORIARTY, K. *et al.* **PKCS# 1: RSA cryptography specifications version 2.2**. [S.l.], 2016.

NAKAMOTO, S. Bitcoin whitepaper. **URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)**, 2008.

NARAYANAN, A. *et al.* **Bitcoin and Cryptocurrency Technologies**. Princeton University Press, 2016. Acessado em 11 de outubro de 2025. Disponível em: <https://press.princeton.edu/books/hardcover/9780691171692/bitcoin-and-cryptocurrency-technologies>.

NPM, INC. **npm Docs**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://docs.npmjs.com/>.

OLIVEIRA, R. R. Criptografia simétrica e assimétrica-os principais algoritmos de cifragem. **Segurança Digital [Revista online]**, v. 31, p. 11–15, 2012.

OPENJS FOUNDATION. **Node.js Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://nodejs.org/en/docs>.

PALMA, L. M. *et al.* Blockchain and smart contracts for higher education registry in brazil. **International Journal of Network Management**, Wiley Online Library, 2019.

PEDERSEN, A. B. *et al.* A ten-step decision path to determine when to use blockchain technologies. **MIS Quarterly Executive**, Indiana University, v. 18, n. 2, p. 99–115, 2019.

PEREIRA, F. L. *et al.* A importância da inovação na gestão de processos administrativos da universidade pública, por meio da implementação da tecnologia de certificação digital. **Revista da UNIFEBE**, v. 1, n. 21, p. 1–23, 2017.

PEREIRA, R. R. Estudo de caso sobre a tecnologia blockchain, projeto ethereum e viabilidade de métodos de mineração. **Ciência da Computação-Tubarão**, 2018.

Polícia Federal. **PF deflagra operação contra falsificação de diplomas em Rondônia e Roraima**. 2025. Acessado em 06 de outubro de 2025. Disponível em: <https://www.gov.br/pf/pt-br/assuntos/noticias/2025/08/pf-deflagra-operacao-contra-falsificacao-de-diplomas-em-rondonia-e-roraima>.

REMIX SOFTWARE. **React Router Docs**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://reactrouter.com/>.

SAAD, S. M. S.; RADZI, R. Z. R. M. Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos). **International Journal of Innovative Computing**, v. 10, n. 2, 2020.

SANTOLIM, C. Análise econômica da cybersegurança aplicada à blockchain. **Revista Jurídica Luso-Brasileira**, p. 863–877, 2020.

SANTOS, R. **Juíza condena envolvidos no desvio de recursos captados pela Gama Filho**. Consutor Jurídico, 2023. Acessado em 04 de outubro de 2023. Disponível em: <https://www.conjur.com.br/2023-mai-05/juiza-condena-reus-desvio-recursos-captados-gama-filho>.

SMOLENSKI, N.; HUGHES, D. Academic credentials in an era of digital decentralization. **Learning Machine Research**, 2016.

SOLIDITY. **Solidity Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://docs.soliditylang.org/>.

STALLINGS, W. **Cryptography and Network Security: Principles and Practice**. 8th. ed. [S.l.]: Pearson, 2022.

SZABO, N. Smart contracts: Building blocks for digital free markets (c) 1995. **Extropy Journal of Transhuman Thought**, 1996.

TRUFFLE. **Truffle Suite Documentation**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://trufflesuite.com/docs/>.

TRUFFLE SUITE. **Ganache**. 2025. Acessado em 11 de outubro de 2025. Disponível em: <https://trufflesuite.com/ganache/>.

University of Nicosia. **University of Nicosia is the first university in the world to publish academic certificates on the blockchain**. 2020. Acessado em 11 de outubro de 2025. Disponível em: <https://abre.ai/unicac>.

ZHENG, Z. *et al.* **An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends**, p. 2, 06 2017.

ZYSKIND, G.; NATHAN, O.; PENTLAND, A. Decentralizing privacy: Using blockchain to protect personal data. In: **2015 IEEE Security and Privacy Workshops (SPW)**. [S.l.: s.n.], 2015. p. 180–184.

APÊNDICE A

Guia de instalações e comandos para executar localmente o DApp.

O fluxo de desenvolvimento típico é: editar código em *Visual Studio Code*, compilar e migrar contratos com *Truffle* para a *blockchain* local fornecida pelo *Ganache*, e executar o frontend *React* que consome os *artifacts* (*JSON* com *ABI* e *bytecode*).

Para que o sistema desenvolvido funcione corretamente, é necessário preparar o ambiente de execução no Windows instalando todas as ferramentas de suporte. O primeiro passo consiste em instalar o *Node.js*, que fornece o interpretador *JavaScript* e o gerenciador de pacotes *npm*, utilizados tanto pelo *Truffle* quanto pela aplicação *React*. Essa instalação pode ser feita de duas formas: acessando o site oficial (https://nodejs.org) e baixando o instalador da versão *LTS*, ou utilizando o *nvm-windows*, ferramenta que permite gerenciar múltiplas versões do *Node*. Após a instalação, pode-se verificar se o ambiente está ativo executando os comandos *node* -v e *npm* -v no *PowerShell*.

Em seguida, é preciso instalar o *Ganache*, que funciona como uma *blockchain* local para testes e simulações. Existem duas opções: a versão com interface gráfica, disponível no site oficial da *Truffle* Suite, em que basta abrir e clicar em *Quickstart* para gerar uma rede privada com *RPC* em http://127.0.0.1:7545; ou a versão de linha de comando, instalada com o comando *npm install -g ganache*, que pode ser executada com *ganache* –port 7545 –chainId 1337 –networkId 1337.

Com o ambiente de *blockchain* configurado, instala-se o *Truffle*, responsável pela compilação, migração e gerenciamento dos contratos inteligentes. Ele pode ser utilizado diretamente via *npx*, sem instalação global, ou instalado com *npm install -g truffle*. O *Truffle* gera os artefatos necessários para a comunicação entre o frontend e os contratos, como a *ABI* e o *bytecode*.

Para a interação entre a interface e a *blockchain*, é essencial instalar a extensão *MetaMask* no navegador. Ela funciona como uma carteira digital que conecta o navegador ao provedor *Web3*, permitindo autorizar transações, alternar contas e redes.

Sua instalação é feita acessando o site oficial (https://metamask.io) e adicionando a extensão ao navegador desejado.

Como ambiente de desenvolvimento, recomenda-se utilizar o *Visual Studio Code*, que pode ser instalado pelo site (https://code.visualstudio.com) ou pelo comando *winget install –id Microsoft.VisualStudioCode* no *PowerShell*. Após abrir o projeto no *VS Code*, deve-se instalar as dependências com *npm install* na raiz e também dentro da pasta do frontend (*my-dapp-1*).

Com tudo configurado, executam-se os comandos *npx truffle compile* para compilar os contratos e *npx truffle migrate –network development* para implantá-los na rede local do *Ganache*. Após a migração, os arquivos *JSON* com as informações dos contratos são gerados automaticamente dentro de *my-dapp-1/src/abi*, permitindo que a interface reconheça os contratos implantados.

Por fim, o frontend é inicializado com o comando *npm start* dentro da pasta *my-dapp-1*. Isso faz com que a aplicação *React* seja executada em *http://localhost:3000*, comunicando-se com o *Ganache* e com a *MetaMask* para realizar todas as operações do sistema.

APÊNDICE B

Contrato de Autoridade:

Listagem 5.0.1: Arquivo completo do contrato de *Autoridade*

```
// SPDX-License-Identifier: MIT
// Autoridade.sol
pragma solidity ^0.8.0;

import "./Estruturas.sol";

// Contrato Autoridade para gerenciar certificados
contract Autoridade {

    address private dono; // Proprietário do contrato
    address[] private revogados; // Lista de endereços de certificados
        revogados

    // Modificador para permitir apenas que o proprietário execute
        certas funções
    modifier somenteDono {
        require(msg.sender == dono, "Somente o proprietario pode chamar
            esta funcao");
        _;
    }

    // Mapeamento de endereços para certificados
    mapping (address => Estruturas.Certificado) private certificados;

    // Construtor que define o proprietário do contrato como o criador
        do contrato
    constructor() {
        dono = msg.sender;
    }
```

```
26
27      // Evento emitido quando um certificado éadicionado
28      // (inclui o nome para facilitar indexacao off-chain)
29      event CertificadoEmitido(address de, address para, uint data, string
            nome);
30
31      // Função para adicionar um certificado
32      function adicionarCertificadoInstituicao(
33          address instituicao,
34          string calldata nomeInstituicao,
35          uint validade
36      ) public somenteDono {
37          require(instituicao != address(0), "Endereco da instituicao
                invalido");
38
39          certificados[instituicao].nome = nomeInstituicao;
40          certificados[instituicao].validade = validade;
41          certificados[instituicao].revogado = false;
42          certificados[instituicao].registrado = true;
43
44          emit CertificadoEmitido(dono, instituicao, block.timestamp,
                nomeInstituicao);
45      }
46
47      // Evento emitido quando um certificado érevogado
48      event CertificadoRevogado(address de, address para, uint data);
49
50      // Função para revogar um certificado
51      function revogarCertificadoInstituicao(address enderecoInstituicao)
            public somenteDono {
52          require(
53              enderecoInstituicao != address(0) && certificados[
                    enderecoInstituicao].registrado,
54              "Endereco do certificado invalido ou nao registrado"
55          );
56
57          certificados[enderecoInstituicao].revogado = true;
```

```
58          revogados.push(enderecoInstituicao);
59
60          emit CertificadoRevogado(dono, enderecoInstituicao, block.
                timestamp);
61      }
62
63      // Função para verificar se um certificado éválido
64      function verificarCertificadoInstituicao(address instituicao) public
             view returns (bool) {
65          require(
66              instituicao != address(0) && certificados[instituicao].
                    registrado,
67              "Instituicao invalida ou nao registrada"
68          );
69
70          if (certificados[instituicao].revogado || certificados[
                instituicao].validade < block.timestamp) {
71              return false;
72          }
73          return true;
74      }
75
76      // retorna o nome da instituicao
77      function nomeDaInstituicao(address instituicao) external view
            returns (string memory) {
78          return certificados[instituicao].nome;
79      }
80
81      // Getter completo
82      function getCertificadoInstituicao(address instituicao)
83          external
84          view
85          returns (string memory nome, uint validade, bool revogado, bool
                registrado)
86      {
87          Estruturas.Certificado storage c = certificados[instituicao];
88          return (c.nome, c.validade, c.revogado, c.registrado);
```

```
89      }
90
91      // Função para obter a lista de endereços de certificados revogados
92      function InstituicoesRevogadas() external view returns (address[]
            memory) {
93          return revogados;
94      }
95  }
```

Contrato de Currículo:

Listagem 5.0.2: Arquivo completo do contrato de *Currículo*

```
1
2   // SPDX-License-Identifier: MIT
3   pragma solidity ^0.8.0;
4
5   import "./Autoridade.sol";
6   import "./Estruturas.sol";
7
8   contract Curriculo {
9       address private dono;
10      Autoridade private autoridade;
11      uint private cargaMinima;
12
13      modifier somenteDono {
14          require(msg.sender == dono, "Somente o proprietario pode chamar
                esta funcao");
15          _;
16      }
17
18      // Qualquer instituicao certificada
19      modifier somenteInstituicaoAutorizada {
20          require(autoridade.verificarCertificadoInstituicao(msg.sender),
                "Instituicao nao autorizada");
21          _;
22      }
23
24      // Somente a instituicao que registrou este aluno
```

```
25    modifier somenteInstituicaoDoAluno(address estudante) {
26        require(instituicaoDoAlunoMap[estudante] == msg.sender, "Somente
              a instituicao do aluno");
27        _;
28    }
29
30    mapping(address => Estruturas.Estudante) private estudantes;
31    mapping(bytes32 => Estruturas.Disciplina) private disciplinas;
32    mapping(address => bool) private diplomaPronto;
33
34    // aluno -> instituicao que o registrou
35    mapping(address => address) private instituicaoDoAlunoMap;
36
37    constructor(uint DisciplinasACumprir, address ContratoAutoridade,
          address instituicaoAdmin) {
38        require(ContratoAutoridade != address(0), "Endereco da
              autoridade invalido");
39        dono = instituicaoAdmin;
40        autoridade = Autoridade(ContratoAutoridade);
41        cargaMinima = DisciplinasACumprir;
42    }
43
44    event EstudanteRegistrado(address indexed instituicao, address
          indexed estudante, uint data);
45    event DiplomaPronto(address indexed estudante);
46
47    function registrarEstudante(address estudante, string memory nome,
          string memory curso)
48        public
49        somenteInstituicaoAutorizada
50    {
51        require(estudante != address(0) && !estudantes[estudante].
              registrado,
52                "Endereco do estudante invalido ou ja registrado");
53
54        estudantes[estudante].nome = nome;
55        estudantes[estudante].registrado = true;
```

```
56          estudantes[estudante].concluidas = 0;
57          estudantes[estudante].curso = curso;
58          diplomaPronto[estudante] = false;
59
60          // guarda quem registrou
61          instituicaoDoAlunoMap[estudante] = msg.sender;
62
63          emit EstudanteRegistrado(msg.sender, estudante, block.timestamp)
                ;
64      }
65
66      function registrarDisciplina(bytes32 id, string memory nome, uint
            cargaHoraria)
67          public
68          somenteInstituicaoAutorizada
69      {
70          require(!disciplinas[id].registrada, "Disciplina ja registrada")
                ;
71          disciplinas[id].registrada = true;
72          disciplinas[id].nome = nome;
73          disciplinas[id].duracao = cargaHoraria;
74      }
75
76      // somente a instituicao que registrou o aluno pode marcar/retirar
            conclusao
77      function disciplinaConcluida(bytes32 disciplina, bool status,
            address estudante)
78          public
79          somenteInstituicaoAutorizada
80          somenteInstituicaoDoAluno(estudante)
81      {
82          require(
83              estudante != address(0) &&
84              estudantes[estudante].registrado &&
85              disciplinas[disciplina].registrada,
86              "Estudante ou disciplina invalidos ou nao registrados"
87          );
```

```
88
89          bool anterior = estudantes[estudante].creditos[disciplina].
                concluida;
90          estudantes[estudante].creditos[disciplina].concluida = status;
91
92          if (!anterior && status) {
93              estudantes[estudante].concluidas += 1;
94          } else if (anterior && !status && estudantes[estudante].
                concluidas > 0) {
95              estudantes[estudante].concluidas -= 1;
96          }
97
98          if (estudantes[estudante].concluidas >= cargaMinima) {
99              diplomaPronto[estudante] = true;
100             emit DiplomaPronto(estudante);
101         } else {
102             diplomaPronto[estudante] = false;
103         }
104     }
105
106     function alunoAptoAGraduar(address estudante) public view returns (
            bool) {
107         require(estudante != address(0), "Endereco de estudante invalido
                ");
108         return diplomaPronto[estudante];
109     }
110
111     // usado pelo Diploma para checar titularidade academica
112     function instituicaoDoAluno(address estudante) external view returns
             (address) {
113         return instituicaoDoAlunoMap[estudante];
114     }
115
116     // Admin
117     function setCargaMinima(uint novaCarga) external somenteDono {
            cargaMinima = novaCarga; }
118     function getCargaMinima() external view returns (uint) { return
```

```
        cargaMinima; }
```

Contrato de Diploma:

Listagem 5.0.3: Arquivo completo do contrato de *Diploma*

```
1
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  import "./Estruturas.sol";
6
7  // Interfaces mínimas
8  interface ICurriculo {
9      function alunoAptoAGraduar(address estudante) external view returns
           (bool);
10     function instituicaoDoAluno(address estudante) external view returns
           (address);
11 }
12
13 interface IAutoridade {
14     function verificarCertificadoInstituicao(address who) external view
           returns (bool);
15 }
16
17 contract Diploma {
18     address private concedente; // instituicao emissora (quem registrou
           o aluno)
19     address private dono; // aluno
20
21     ICurriculo private curriculo;
22     IAutoridade private autoridade;
23
24     Estruturas.Documento private diploma;
25
26     modifier somenteConcedente() {
27         require(msg.sender == concedente, "Somente o concedente");
28         _;
29     }
```

```
30
31      modifier somenteConcedenteCertificada() {
32          require(msg.sender == concedente, "Somente o concedente");
33          require(autoridade.verificarCertificadoInstituicao(msg.sender),
                "Instituicao nao autorizada");
34          _;
35      }
36
37      /// @param estudante dono do diploma
38      /// @param instituicao emissor/concedente (deve ser msg.sender)
39      /// @param contratoCurriculo endereco do Curriculo
40      /// @param contratoAutoridade endereco da Autoridade
41      constructor(
42          address estudante,
43          address instituicao,
44          address contratoCurriculo,
45          address contratoAutoridade
46      ) {
47          require(
48              estudante != address(0) &&
49              instituicao != address(0) &&
50              contratoCurriculo != address(0) &&
51              contratoAutoridade != address(0),
52              "Endereco invalido"
53          );
54
55          require(instituicao == msg.sender, "Emissor deve ser msg.sender
                ");
56
57          curriculo = ICurriculo(contratoCurriculo);
58          autoridade = IAutoridade(contratoAutoridade);
59
60          require(autoridade.verificarCertificadoInstituicao(msg.sender),
                "Deploy bloqueado: instituicao nao autorizada");
61          require(curriculo.instituicaoDoAluno(estudante) == instituicao,
                "Aluno nao pertence a esta instituicao");
62
```

```
63          diploma.receptor = estudante;
64          diploma.emissor = instituicao;
65
66          dono = estudante;
67          concedente = instituicao;
68      }
69
70      event DiplomaEmitido(address indexed de, address indexed para,
            uint256 data);
71      event DiplomaRevogado(address indexed de, uint256 data);
72
73      function emitirDiploma() public somenteConcedenteCertificada {
74          require(!diploma.status, "Ja emitido");
75          require(curriculo.instituicaoDoAluno(dono) == concedente, "Aluno
                nao pertence a esta instituicao");
76          require(curriculo.alunoAptoAGraduar(dono), "O estudante nao
               concluiu todas as disciplinas");
77
78          diploma.data = block.timestamp;
79          diploma.status = true;
80
81          emit DiplomaEmitido(diploma.emissor, diploma.receptor, diploma.
               data);
82      }
83
84      function revogarDiploma() public somenteConcedenteCertificada {
85          require(diploma.status, "Nao emitido ou ja revogado");
86          require(curriculo.instituicaoDoAluno(dono) == concedente, "Aluno
                nao pertence a esta instituicao");
87
88          diploma.status = false;
89          emit DiplomaRevogado(concedente, block.timestamp);
90      }
91
92      function diplomaValido() public view returns (bool) {
93          return diploma.status;
94      }
```

```
95
96      // Getters úteis (opcionais)
97      function getConcedente() external view returns (address) { return
            concedente; }
98      function getAluno() external view returns (address) { return dono; }
99  }
```

Contrato de Estruturas:

Listagem 5.0.4: Arquivo completo do contrato de *Estruturas*

```
1
2   // SPDX-License-Identifier: MIT
3   // Estruturas.sol
4   pragma solidity ^0.8.0;
5
6   library Estruturas {
7       struct Certificado {
8           string nome;
9           uint validade; // Data de expiração do certificado da instituiçã
                o
10          bool revogado; // Indica se o certificado da instituição foi
                revogado
11          bool registrado; // Indica se o certificado da insituição está
                registrado
12      }
13
14      struct Disciplina {
15          bool concluida; // Indica se a disciplina foi concluída
16          bool registrada; // Indica se a disciplina está registrada
17          uint duracao; // Duração da disciplina
18          string nome; // Nome da disciplina
19
20      }
21
22      struct Estudante {
23          string nome; // Nome do estudante
24          bool registrado; // Indica se o estudante está registrado
25          uint concluidas; // Número de disciplinas concluídas
```

```
26          string curso; // Nome do curso que o estudando está matriculado
27          mapping(bytes32 => Disciplina) creditos; // Mapeamento de
                disciplinas para o estudante
28      }
29
30      struct Documento {
31          address emissor; // Emissor do documento
32          address receptor; // Receptor do documento
33          bytes32 titulo; // Título do documento
34          bytes32 descricao; // Descrição do documento
35          uint data; // Data do documento
36          bool status; // Status do documento
37      }
38  }
```