UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RIAN BORGES BARBOSA

A APLICAÇÃO DE *SMART CONTRACTS* PARA O COMPARTILHAMENTO DE DADOS EM *HEALTHCARE*

CHAPECÓ
2026

RIAN BORGES BARBOSA

A APLICAÇÃO DE *SMART CONTRACTS* PARA O COMPARTILHAMENTO DE DADOS EM *HEALTHCARE*

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal da Fronteira Sul (UFFS), como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Geomar André Schreiner

CHAPECÓ
2026

Barbosa, Rian Borges

A APLICAÇÃO DE *SMART CONTRACTS* PARA O COMPARTILHAMENTO DE DADOS EM *HEALTHCARE* / Rian Borges Barbosa – 2026.

71 f.

Orientador: Geomar André Schreiner

Trabalho de Conclusão de Curso (Graduacão) – Universidade Federal da Fronteira Sul, Curso de Ciência da Computação, Chapecó, SC, 2026.

1. Blockchain 2. Smart Contracts 3. Dados de Saúde 4. Segurança I. , Geomar André Schreiner, orient. II. Universidade Federal da Fronteira Sul. III. Título.

"Science is but a perversion of itself unless it has as its ultimate goal the betterment of humanity."
(Nikola Tesla)

AGRADECIMENTOS

A meus pais, minha família, pelo apoio e ajuda durante toda a minha trajetória acadêmica, muito obrigado, sem eles eu não teria chegado onde estou. Também agradeço a todo o corpo docente da Universidade Federal da Fronteira Sul - UFFS pelo ensinamento e dedicação. Em especial, agradeço ao professor doutor Geomar Schreiner, cuja orientação foi fundamental para eu chegar até aqui.

RESUMO

Nos últimos anos, a área médica tem disposto de uma base de dados centralizada, um repositório para a unificação de dados de saúde, como registros de pacientes. No entanto, esta proposta apresenta desafios sistêmicos de segurança por se tratar de dados sigilosos. A unificação desses dados através de uma rede descentralizada solucionaria esse problema e facilitaria a gestão de dados em grande escala. Este trabalho propõe a utilização de *smart contracts* juntamente à rede *blockchain* para a centralização e compartilhamento de dados de saúde. A aplicação desenvolvida mostrou-se viável, demonstrando que é possível garantir a integridade dos dados e o controle de acesso aos dados.

Palavras-Chave: Blockchain, Smart Contracts, Dados de Saúde, Segurança.

ABSTRACT

In recent years, the medical field has relied on a centralized database, a repository for unifying health data such as patient records. However, this proposal presents systemic security challenges, as it involves confidential information. Unifying this data through a decentralized network would address these issues and facilitate large-scale data management. This work proposes the use of smart contracts in conjunction with blockchain technology for the centralization and sharing of health data. The developed application proved to be viable, demonstrating that it is possible to ensure data integrity and control access to the information.

Keywords: Blockchain, Smart Contracts, Health Data, Security.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

LISTA DE SIGLAS

POW – Proof of Work

POS – Proof of Stake

API – Application Programming Interface

IPFS – InterPlanetary File System

P2P – Peer-to-Peer

EVM – Ethereum Virtual Machine

LIFO – Last-In-First-Out

CID – Content Identifier

PBKDF2 – Password-Based Key Derivation Function 2

DAPP – Decentralized Application

RBAC – Role-Based Access Control

GDPR – General Data Protection Regulation

AES – Advanced Encryption Standard

DEFI – Decentralized Finance

1. INTRODUÇÃO

A transformação digital na área da saúde tem promovido mudanças significativas na forma como os dados são coletados, armazenados e compartilhados. O compartilhamento de informações clínicas entre diferentes instituições, por meio de prontuários eletrônicos do paciente (PEP) e mecanismos de troca de informações em saúde (*Health Information Exchange*), tem se consolidado como elemento central da gestão dessas informações (WANG et al., 2024). A diversidade de entidades médicas, desde unidades básicas de saúde até hospitais especializados, impulsionou o desenvolvimento de novas soluções tecnológicas voltadas para a centralização e integração de dados de pacientes. Contudo, a gestão de dados de saúde demanda atenção especial no que se refere à privacidade e à segurança das informações. A natureza sensível dos dados médicos, que incluem históricos clínicos, resultados de exames, informações genéticas e dados pessoais, exige a preservação do sigilo médico-paciente. Simultaneamente, a crescente necessidade de compartilhamento de informações entre diferentes profissionais de saúde e instituições requer sistemas com arquiteturas mais robustas e seguras para atender às demandas do cenário atual (SHOJAEI; VLAHU-GJORGIEVSKA; CHOW, 2024).

Nesse contexto, emerge uma abordagem que busca resolver os desafios de interoperabilidade na área da saúde: a utilização de tecnologias *blockchain*. Esta tecnologia, que já demonstrou eficácia na solução de problemas complexos em diversos setores, apresenta características únicas que a tornam particularmente adequada para o ambiente de saúde (HALEEM et al., 2021). A *blockchain*, com suas propriedades fundamentais de descentralização, imutabilidade, transparência e segurança criptográfica, oferece uma plataforma para o gerenciamento e compartilhamento seguro de dados médicos sensíveis (SHOJAEI; VLAHU-GJORGIEVSKA; CHOW, 2024). Suas características permitem criar um ambiente confiável onde múltiplas partes podem interagir sem a necessidade de intermediários centralizados, mantendo a integridade e a rastreabilidade das informações.

Os dados médicos são altamente sensíveis e sua proteção representa uma prioridade absoluta. Os sistemas tradicionais de gestão, caracterizados pela dependência de intermediários e pela arquitetura centralizada, são particularmente suscetíveis a ataques cibernéticos. A tecnologia *blockchain*, com sua estrutura descentralizada e propriedades de imutabilidade, oferece uma camada adicional de segurança contra adulterações de dados e acessos não autorizados (SHOJAEI; VLAHU-GJORGIEVSKA; CHOW, 2024). A natureza transparente e auditável da tecnologia *blockchain* pode fortalecer significativamente a confiança entre pacientes e profissionais da saúde, pois cada transação na *blockchain* é verificável e rastreável, proporcionando elevada transparência e rastreabilidade ao sistema (HALEEM et al., 2021).

Os processos burocráticos tradicionalmente empregados na troca de informações médicas, caracterizados pela dependência de múltiplos intermediários, resultam em atrasos substanciais e custos operacionais elevados. A implementação dos contratos inteligentes permite a automação de processos complexos, reduzindo significativamente o tempo de processamento e os custos associados (HALEEM et al., 2021). Essa automação não só melhora a eficiência operacional, mas também minimiza a probabilidade de erros humanos, um fator crítico em ambientes médicos onde a precisão é vital. A redução desses intermediários resulta em menor dispêndio de recursos com terceiros e tarefas burocráticas, permitindo que as instituições direcionem maiores investimentos para o atendimento direto ao paciente e a melhoria da qualidade assistencial.

Diante deste cenário, este trabalho tem como objetivo principal explorar a aplicação da tecnologia *blockchain* e dos contratos inteligentes (*smart contracts*) no âmbito da gestão de dados médicos.

Para alcançar o objetivo, foi desenvolvida uma solução baseada em tecnologia *blockchain* e contratos inteligentes para a gestão de dados médicos. A implementação da solução busca automatizar processos burocráticos, reduzir custos operacionais e eliminar intermediários desnecessários no fluxo de dados médicos. Por meio das características da *blockchain* como a descentralização, imutabilidade e transparência, o sistema visa minimizar vulnerabilidades comuns em sistemas centralizados, como acessos não autorizados e violações de privacidade. A aplicação desenvolvida oferece controle de acesso granular às informações médicas, permitindo que várias partes interessadas interajam de forma segura, enquanto mantém a conformidade com as regulamentações de proteção de dados.

Os experimentos foram conduzidos em um ambiente local controlado, no qual foram utilizados a *blockchain* Ethereum local (Ganache), o *framework* Truffle para a compilação e execução dos testes e a implementação em Javascript do protocolo IPFS para o armazenamento distribuído de arquivos. Os resultados demonstraram que o sistema atende aos requisitos estabelecidos: a imutabilidade dos registros foi comprovada por meio da análise da estrutura de blocos da *blockchain*, o controle de acesso granular operou conforme especificado pelos *smart contracts*, e a integração com a IPFS, combinada com a criptografia `AES-256-GCM` aplicada previamente ao armazenamento, mostrou-se eficaz para a proteção de arquivos médicos sensíveis. A rastreabilidade completa das operações foi validada por meio dos identificadores criptográficos e eventos emitidos pelos contratos, confirmando a viabilidade técnica da solução proposta para a gestão segura e transparente de dados de saúde.

Considerando o potencial da tecnologia *blockchain* e dos contratos inteligentes para resolver problemas críticos na gestão de dados médicos, este estudo justifica-se tanto por sua aplicabilidade prática quanto pela contribuição acadêmica. A exploração dessas tecnologias emergentes contribui para a inovação contínua no campo da saúde digital,

oferecendo fundamentos sólidos para futuras pesquisas e o desenvolvimento de aplicações especializadas. Além disso, a pesquisa atende às demandas crescentes por soluções tecnológicas que conciliem eficiência operacional, segurança de dados e conformidade regulatória no setor de saúde. Considerados em conjunto, esses fatores demonstram a necessidade urgente de investigar e desenvolver soluções baseadas em *blockchain* para a gestão de dados médicos, posicionando este trabalho como uma contribuição relevante para o avanço científico e tecnológico na área de saúde digital.

1.1 Organização do Texto

Este trabalho está estruturado em sete capítulos, que abordam desde os fundamentos teóricos da tecnologia *blockchain* até a apresentação da solução proposta para o compartilhamento seguro de dados em *healthcare*.

No Capítulo 2, são explorados os conceitos fundamentais da tecnologia *blockchain*, detalhando suas arquiteturas de rede, *smart contracts* e suas aplicações no contexto de *healthcare*. O Capítulo 3 apresenta uma revisão dos trabalhos relacionados, analisando diferentes abordagens para a aplicação de *blockchain* e *smart contracts* no gerenciamento de dados de saúde.

O Capítulo 4 apresenta a metodologia adotada, detalhando os critérios de seleção da plataforma *blockchain*, as ferramentas de desenvolvimento e os aspectos de segurança considerados.

O Capítulo 5 aborda a aplicação *BlockHealth* desenvolvida, apresentando sua implementação, uma visão geral da arquitetura e a estrutura técnica.

O Capítulo 6 apresenta a avaliação da solução desenvolvida, incluindo os resultados obtidos quanto ao armazenamento seguro dos dados, a interação com o protocolo IPFS (*InterPlanetary File System*) e aos pontos de melhoria identificados.

Por fim, no Capítulo 7, são apresentadas as conclusões, discutindo os resultados alcançados, as limitações identificadas e as possibilidades de trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

O propósito desta pesquisa é o estudo e a aplicação de *smart contracts* na *blockchain* para o compartilhamento de dados na área da saúde. Nos últimos anos, com a revolução tecnológica proporcionada pelo desenvolvimento da *blockchain*, diversas áreas e indústrias foram transformadas, inclusive a área da saúde. Este estudo descreve como a aplicação dos *smart contracts* podem ser aplicados para otimizar o compartilhamento seguro e eficiente de dados médicos.

Nas seções seguintes, será explorado o conceito de *blockchain* e sua importância, os fundamentos dos contratos inteligentes e como eles podem ser aplicados ao uso de dados médicos sensíveis na rede *blockchain*; será abordada uma forma prática de seu funcionamento, com o objetivo de descrever a segurança e os benefícios que a tecnologia proporciona.

2.1 Conceito da Tecnologia Blockchain

De acordo com (ROUHANI; DETERS, 2019), podemos definir *blockchain* como uma tecnologia de contabilidade distribuída confiável que realiza a replicação e o compartilhamento de dados entre redes *peer-to-peer*, construída sobre registros ou blocos que atuam como um banco de dados. Embora as bases conceituais dessa tecnologia tenham sido estabelecidas no final da década de 1970, o termo *blockchain* foi formalmente introduzido em 2008 através do artigo de Satoshi Nakamoto intitulado *"Bitcoin: A Peer-to-Peer Electronic Cash System"* (NAKAMOTO, 2008), no qual o autor apresenta a utilização dessa tecnologia para transações de bitcoin em uma plataforma online de pagamentos, sem a necessidade de intermediação de instituições financeiras.

A tecnologia *blockchain* é caracterizada por uma cadeia de blocos na qual cada bloco contém informações armazenadas sobre transações e dados do sistema (KHATOON, 2020). Cada bloco tem a responsabilidade de registrar, validar e distribuir transações para outros nós da rede. A integridade da cadeia é assegurada pelo fato de que a remoção ou alteração de qualquer bloco comprometeria a validade de todos os blocos subsequentes, criando um sistema intrinsecamente seguro contra adulterações.

Uma *blockchain*, de modo geral, comporta-se como um banco de dados distribuído e funciona como um "livro-razão"(*ledger*) digital, sendo caracterizada como uma tecnologia imutável, descentralizada e compartilhada. Sua natureza descentralizada decorre do fato de operar em redes *peer-to-peer*, não dependendo de uma autoridade central ou de um servidor único para seu funcionamento. Conforme explicado por (KHATOON,

2020), um grupo de transações em redes *blockchain* é combinado em blocos, que são ligados à cadeia utilizando o *hash* criptográfico do bloco anterior.

A Figura 2.1 ilustra uma estrutura *blockchain*, tendo como ponto de partida o bloco gênesis, que armazena o *hash* de seu próprio conteúdo. Cada bloco subsequente contém o *hash* de seu próprio conteúdo, o *hash* do bloco anterior e suas respectivas transações. Dessa forma, as informações são distribuídas em uma rede de computadores denominados nós, operando de forma independente e garantindo a redundância e segurança dos dados armazenados.

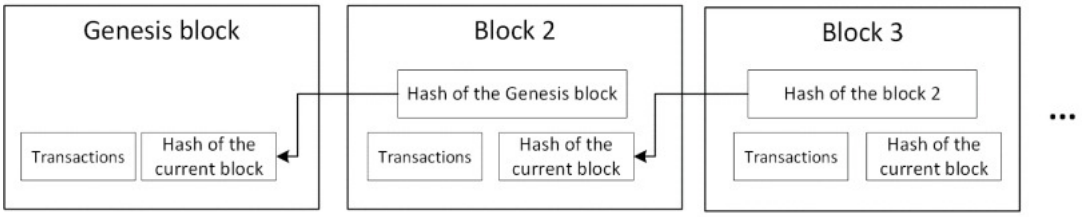


Figura 2.1 – Conexão de blocos na *blockchain* (KHATOON, 2020).

Segundo (MOOSAVI et al., 2024), cada bloco é constituído por duas partes principais: os dados do bloco (*block data*), que contêm os registros das transações e contratos inteligentes, e o cabeçalho do bloco (*block header*). O cabeçalho desempenha um papel crucial na validação e possui um tamanho fixo, geralmente de 80 bytes. Conforme ilustrado na Figura 2.2, ele é composto pela versão (4 bytes), que indica as regras de validação, pelo *Merkle Tree Hash* (32 bytes), que representa o valor final derivado de todas as transações, pelo *Parent Block Hash* (32 bytes), correspondente ao *hash* do bloco anterior, além do timestamp (4 bytes), nBits (4 bytes) e o Nonce (4 bytes), utilizados nos processos de mineração e validação.

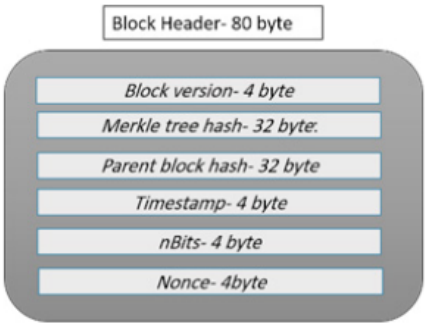


Figura 2.2 – Block Header ((MOOSAVI et al., 2024)).

A Figura 2.3 ilustra como os blocos se conectam em sequência. Cada bloco é composto por um cabeçalho (*Block Header*), que contém o *hash* do bloco anterior (*Preblockhash*) e o *hash* do próprio bloco (*Block hash*), e por uma seção de dados (*Block*

Data), com as transações e os contratos inteligentes. O encadeamento ocorre porque cada bloco referencia o *hash* de seu antecessor, o Bloco k ligando-se ao Bloco $k+1$, o que garante a integridade histórica do registro.

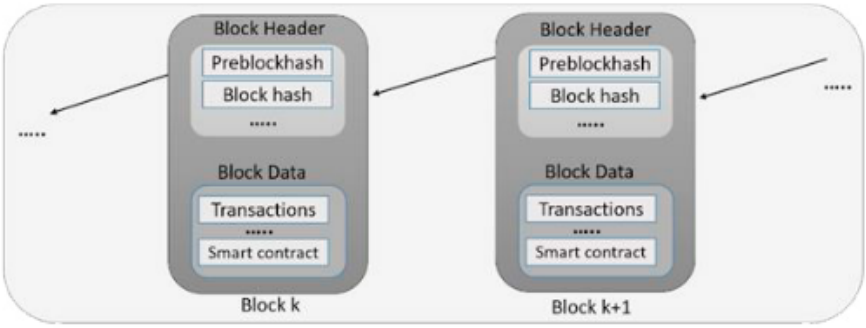


Figura 2.3 – Estrutura do cabeçalho e sequência de blocos na *blockchain* ((MOOSAVI et al., 2024)).

2.2 Conceito de Redes

Para atingir o objetivo proposto, devemos entender o conceito de arquitetura de rede descentralizada e distribuída. Pois redes descentralizadas destacam-se por serem a base tecnológica do sistema *blockchain*, e as redes distribuídas como sua arquitetura representando a forma como os dados são armazenados.

2.2.1 Redes Descentralizadas

As redes descentralizadas são sistemas que distribuem operações entre múltiplos nós, onde cada nó opera independentemente dos outros. Esse modelo é comumente utilizado em sistemas *peer-to-peer* (P2P) e em tecnologias de *blockchain*. A descentralização oferece vários benefícios, principalmente relacionados à segurança, transparência e resiliência das interações dos usuários.

Um dos maiores benefícios é a redução da necessidade de confiança, que é alcançada através de diversos mecanismos: criptografia, que garante que os dados sejam protegidos contra acessos não autorizados; algoritmos de consenso, que facilitam a concordância entre os nós sobre o estado da rede; contratos inteligentes, que permitem a execução automática de acordos codificados em software; e nós independentes, onde cada nó mantém uma cópia do livro-razão, uma lista de registros eletrônicos das transações ao

longo do tempo. Isso ajuda a verificar a consistência das transações e a garantir o controle sobre as interações na rede, prevenindo a degradação da funcionalidade da rede.

De acordo com (SEGAL et al., 2023), os dados em redes descentralizadas são documentados e protegidos de forma contínua em blocos de dados sequenciais. Cada bloco contém todos os dados registrados anteriormente, resultando em uma cadeia de blocos que não pode ser alterada, retroagida ou excluída. Essa característica de imutabilidade é uma das principais vantagens das arquiteturas descentralizadas em comparação com as estruturas de dados gerenciadas por uma única entidade.

A Figura 2.4 aborda a infraestrutura *blockchain*, com sua rede descentralizada, como uma evolução da era pré-internet, da internet e do servidor em nuvem.

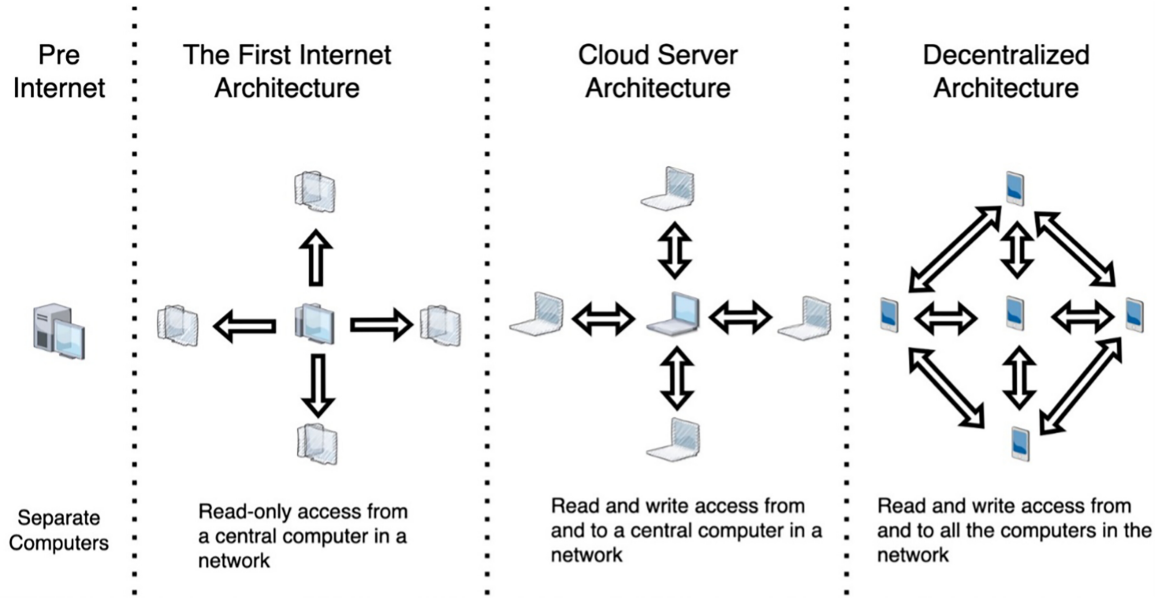


Figura 2.4 – Redes descentralizadas como uma evolução de arquiteturas anteriores (SEGAL et al., 2023).

2.2.2 Redes Distribuídas

Embora relacionadas às redes descentralizadas, as redes distribuídas possuem características técnicas específicas e se baseiam na dispersão de seus componentes e funcionalidades entre múltiplos nós. Um sistema distribuído é "um conjunto de computadores independentes que se apresenta ao usuário como um sistema único e coerente"(TANENBAUM; STEEN, 2016). Trata-se de uma rede inteiramente escalável comparada à descentralizada. A principal diferença reside no fato de que, enquanto redes descentralizadas enfatizam a ausência de uma autoridade central, redes distribuídas concentram-se na distribuição fí-

sica e lógica dos recursos operacionais para melhorar a performance, a tolerância a falhas e a disponibilidade.

No contexto de *blockchain*, as *blockchains* são uma nova abordagem para a criação de redes distribuídas (KHALID et al., 2023), pois representam uma arquitetura onde os dados e operações são distribuídos entre múltiplos nós que, dependendo de seu modelo de consenso e governança, podem alcançar diferentes graus de descentralização. Essa configuração permite que cada nó mantenha a sua própria memória local e estabeleça comunicação exclusivamente através da troca de mensagens, eliminando a necessidade de memória compartilhada centralizada como ilustrado na Figura 2.5.

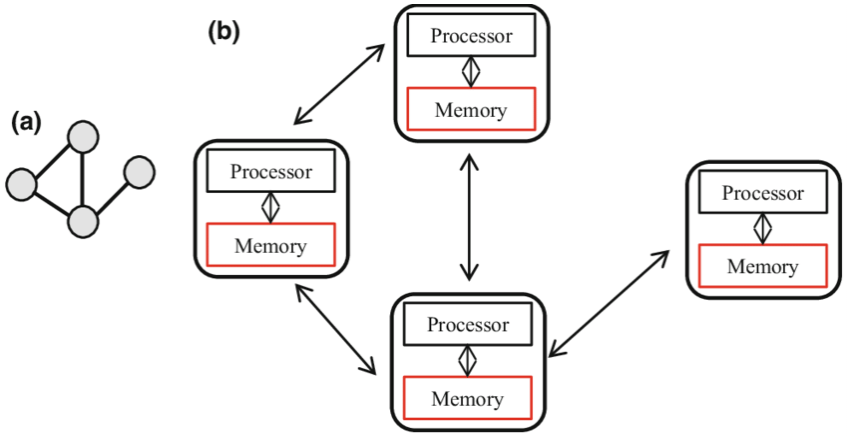


Figura 2.5 – Ilustração de um sistema distribuído típico, onde cada nó possui memória local e interage via troca de mensagens (SAMUEL et al., 2017).

2.3 Ethereum e Seu Processo de Desenvolvimento

A rede *blockchain* possui diversas plataformas, sendo a rede Ethereum uma das mais conhecidas. Grande parte das aplicações descentralizadas, também conhecidas como *DApps* (Decentralized Applications), opera nesta plataforma. A rede Ethereum destaca-se por sua capacidade de executar contratos inteligentes e por ser uma plataforma *open-source*.

Atualmente, a plataforma utiliza o algoritmo de consenso Prova de Participação (*Proof of Stake* - PoS), tendo migrado do algoritmo de Prova de Trabalho (*Proof of Work* - PoW) em setembro de 2022 (Ethereum Foundation, 2022). O algoritmo PoW é um mecanismo utilizado para validar transações e adicionar novos blocos à *blockchain* por meio da mineração (KHATOON, 2020).

A Figura 2.6 representa a execução de um *smart contract* no Ethereum, desde a criação de uma instância até a validação do bloco. Um contrato inteligente contém funções

que podem ser executadas por contas externas. Para executar uma função, a aplicação descentralizada recupera uma instância do contrato inteligente por meio de seu endereço e inicia uma transação. Para que a transação tenha efeito, ela precisa ser processada pelos nós da rede. Após o processamento bem-sucedido, um novo bloco contendo a transação é criado (JAIMAN; UROVI, 2020). Subsequentemente, o bloco é validado, verificado e executado pelos nós pares, sendo então adicionado à *blockchain*.

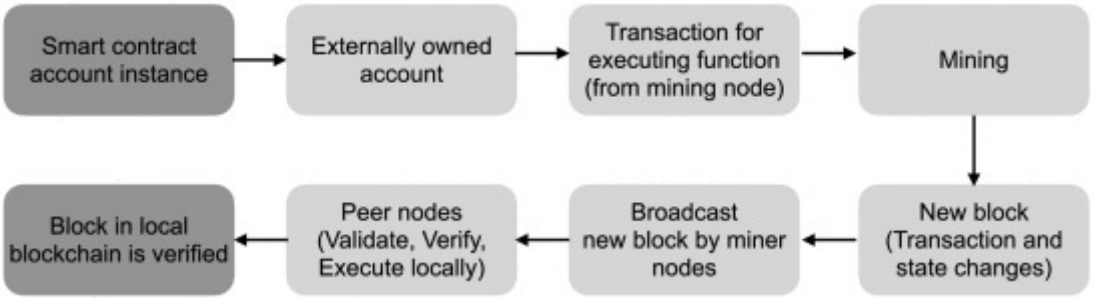


Figura 2.6 – Etapas do processo de execução de um *smart contract* no Ethereum (JAIMAN; UROVI, 2020).

2.4 Smart Contracts

Smart contracts ou contratos inteligentes são programas autoexecutáveis que realizam transações automaticamente quando condições predeterminadas são atendidas. Seu funcionamento baseia-se no "se-então", executando acordos entre as partes de forma autônoma. Esses contratos podem automatizar processos que envolvem regras e cláusulas específicas, sendo armazenados em plataformas *blockchain* como a rede Ethereum.

O conceito de contratos inteligentes foi introduzido por Nick Szabo em 1994 e, posteriormente, implementado por diversos sistemas. Sua popularidade cresceu significativamente após a implementação do Ethereum, o segundo maior sistema baseado em *blockchain*, atraindo intensa investigação sobre suas capacidades. O principal objetivo dos contratos inteligentes é facilitar e executar digitalmente negociações verificáveis entre partes participantes da *blockchain* (ZAGHLOUL; LI; REN, 2019).

A evolução dos contratos inteligentes, desde seu conceito inicial até a implementação prática no Ethereum, demonstra o papel crucial das plataformas *blockchain* na adoção dessas tecnologias. Os *smart contracts* desempenham um papel significativo em diversas indústrias, não apenas agilizando processos, mas também reduzindo a necessidade de intermediários e aumentando a segurança e a transparência das transações.

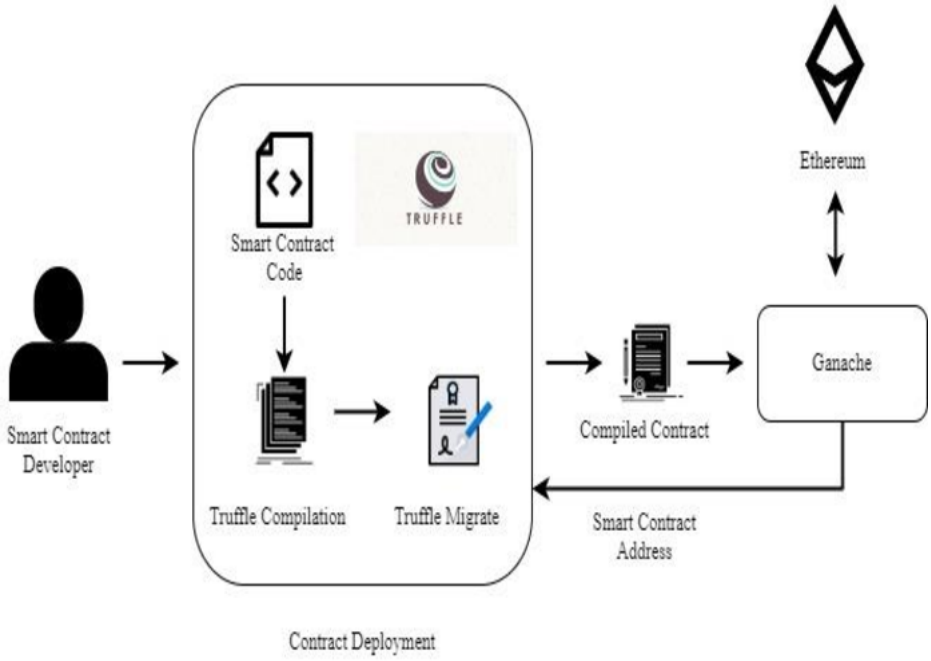


Figura 2.7 – Smart Contract System Architecture (AHAMED, 2023).

A Figura 2.7 apresenta o fluxo de implantação de um contrato inteligente na *blockchain* Ethereum. O desenvolvedor escreve o código do contrato, que é compilado pelo Truffle Compile. O *Truffle Migrate* realiza a implantação do contrato compilado no Ganache, uma *blockchain Ethereum* local para desenvolvimento. Após a implantação, o endereço do contrato inteligente é retornado ao ambiente de desenvolvimento, viabilizando sua utilização pela aplicação. Embora a arquitetura apresentada utilize uma rede de testes para implantação, as etapas de codificação e compilação do contrato inteligente mantêm-se inalteradas para implantações em redes *blockchain* de produção.

2.5 Solidity

Grande parte das *blockchains* possuem uma linguagem ou *script* usado para interagir com a rede. A Ethereum dispõe da linguagem de programação mais popular orientada a contratos, denominada Solidity (CHITTODA, 2019).

Solidity[1] é uma linguagem de alto nível orientada para contratos inteligentes que são executados na *Ethereum Virtual Machine* (EVM). Por ser uma linguagem *Turing-complete*, permite a implementação de estruturas lógicas complexas e algoritmos arbitrários (CHITTODA, 2019). A linguagem abrange desde aplicações financeiras descentralizadas (*DeFi*) a tokens não fungíveis (NFTs).

[1] <https://www.soliditylang.org>

Sua sintaxe é inspirada em linguagens como C++, Python e Javascript (CHITTODA, 2019). Além disso, é estaticamente tipada e oferece recursos avançados como suporte a herança, bibliotecas externas, modificadores, eventos e tipos de dados estruturados (CHITTODA, 2019).

Para a EVM executar estes contratos, primeiro eles são compilados e depois convertidos em *bytecodes* e durante a execução são convertidos em códigos operacionais de máquina (*opcodes*). Ressaltando que a EVM é um interpretador e se baseia em uma pilha onde cada elemento possui 32 bytes, mantendo a consistência com seu sistema de armazenamento em chave-valor que também utiliza palavras de 32 bytes (MUKHOPADHYAY, 2018). A Figura 2.8 mostra a pilha como um processo *LIFO* com suas operações de POP e PUSH.

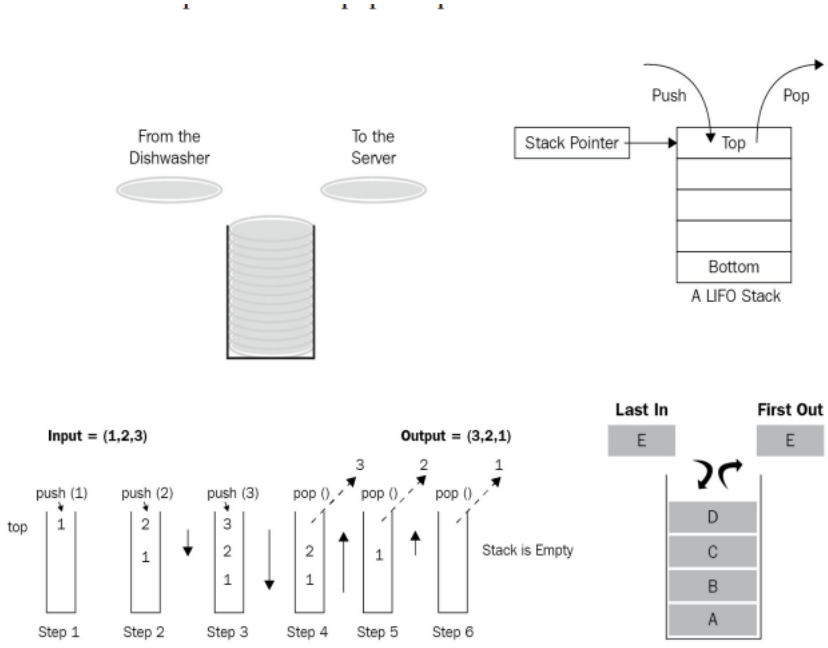


Figura 2.8 – Stack operation LIFO (MUKHOPADHYAY, 2018).

Cada operação de *bytecode* na EVM possui uma taxa associada, denominada *gas*, o preço de cada unidade de gas (*gas price*) é expresso em gwei, sendo 1 gwei igual a 10^{-9} *ether* (Ethereum). Portanto, o *gas* representa o custo total para implantar e executar um contrato inteligente, variando conforme o *opcode* ou instrução, uma vez que cada um possui um custo distinto. Por exemplo, o *opcode* ADD terá um custo de 3 *gas* (YAMAGATA, 2022). A Figura 2.9 abaixo representa os *opcodes* na EVM com os custos associados. A coluna *Initial Stack* indica os operandos que cada *opcode* espera no topo da pilha antes de executar. No ADD, por exemplo, `a` e `b` são consumidos da pilha, somados, e o resultado é devolvido a ela.

O custo de uma transação pode ser expresso em valores monetários a partir da combinação do*gas price*, definido em *gwei*, e a cotação do *ether* no mercado. O custo em *ether* é obtido multiplicando-se o total de *gas* consumido pelo *gas price*. Convertendo esse resultado pela cotação vigente do *ether*, chega-se ao valor em moeda corrente. Uma análise detalhada desse custo, aplicada às operações do sistema desenvolvido, é apresentada na Seção 6.6.

Stack	Name	Gas	Initial Stack
00	STOP	0	
01	ADD	3	a, b
02	MUL	5	a, b
03	SUB	3	a, b
04	DIV	5	a, b
05	SDIV	5	a, b
06	MOD	5	a, b

Figura 2.9 – Opcodes e custos de *gas*. Fonte: www.evm.codes.

2.6 A Rede IPFS

O *InterPlanetary File System* (IPFS) é um protocolo de rede descentralizada e distribuída que opera em arquitetura ponto a ponto (*peer-to-peer*), permitindo o armazenamento e o compartilhamento de arquivos sem depender de servidores centralizados (IPFS Docs, n.d.a). Em vez de localizar os dados por seu endereço em um servidor específico, o IPFS os distribui entre os nós participantes da rede, o que elimina pontos únicos de falha e favorece a redundância e a disponibilidade dos arquivos.

A principal característica do protocolo é o endereçamento baseado em conteúdo (*content-addressing*). Cada arquivo armazenado recebe um identificador único *Content Identifier* (CID), derivado de um *hash* criptográfico calculado a partir de seu próprio conteúdo. Portanto, qualquer alteração no arquivo resulta em um identificador completamente diferente. Essa propriedade permite verificar a integridade dos dados, uma vez que o CID pode ser utilizado para confirmar se o arquivo recuperado corresponde exatamente ao original.

2.7 Compartilhamento de Dados na Healthcare

Conforme a área da saúde vem evoluindo, a necessidade do desenvolvimento tecnológico cresce cada vez mais. Quando se trata da área médica, fala-se muito sobre dados, sejam eles de pacientes ou de dados técnicos. Compartilhar dados para troca de informações, cadastro de pacientes e tudo o que envolve dados críticos requer muita segurança e eficiência.

Os problemas mais significativos enfrentados são a proteção, a partilha e a interoperabilidade dos dados na gestão da saúde da população. A partir deste ponto, a *blockchain* pode favorecer eficientemente esses quesitos específicos em comparação ao compartilhamento de dados em bancos que são usados normalmente. Esta tecnologia melhora a segurança, o intercâmbio de dados, a interoperabilidade, a integridade, a atualização e o acesso em tempo real quando corretamente implementada (HALEEM et al., 2021); ou seja, a *blockchain*, juntamente com *smart contracts*, resolve esses problemas enfrentados na gestão da saúde e, além de melhorar a segurança e outros fatores, os *smart contracts* facilitam o acordo de contratos burocráticos e proporcionam mais confiança tanto para o paciente quanto para a parte prestadora.

A troca de informações de saúde é outro processo demorado e repetitivo que leva a altos custos do setor de saúde, rapidamente resolvido com o uso dessa tecnologia (HALEEM et al., 2021). Com os *smart contracts* na *blockchain*, o processo repetitivo torna-se uma solução fácil, pois esses contratos possuem autoexecução após o acordo entre as duas partes.

A Figura 2.10 apresenta áreas de domínio em que a *blockchain* pode atuar na área médica, desde a proteção de dados até o rastreamento de doenças e surtos. Estas são apenas algumas das áreas em que a tecnologia *blockchain* pode atuar e que envolvem o compartilhamento de dados.

Figura 2.10 – Possibilidades da *blockchain* no domínio da *healthcare* (HALEEM et al., 2021).

3. TRABALHOS RELACIONADOS

Este capítulo apresenta uma análise crítica dos principais trabalhos científicos que abordam a aplicação da tecnologia *blockchain* e contratos inteligentes na gestão de dados de saúde. A revisão sistemática da literatura evidencia o crescente interesse da comunidade acadêmica em soluções descentralizadas para os desafios de segurança, privacidade e interoperabilidade no setor de saúde digital.

Os trabalhos selecionados demonstram diferentes abordagens metodológicas e tecnológicas para a implementação de sistemas baseados em *blockchain* na área da saúde, contribuindo para o embasamento teórico e técnico desta pesquisa. A análise comparativa destes estudos permite identificar lacunas, oportunidades de melhoria e direcionamentos para o desenvolvimento da solução proposta neste trabalho.

3.1 A Blockchain-Based Smart Contract System for Healthcare Management

O trabalho desenvolvido por (KHATOON, 2020) apresenta uma arquitetura abrangente para a gestão de dados de saúde utilizando a tecnologia *blockchain*. A autora propõe um sistema baseado em contratos inteligentes implementado na plataforma Ethereum, com foco na otimização de fluxos de trabalho médicos complexos.

A implementação do sistema envolveu o desenvolvimento de uma aplicação descentralizada (*DApp*) na blockchain Ethereum, utilizando a linguagem de programação Solidity para a criação dos contratos inteligentes. O ambiente de desenvolvimento compreendeu o uso do Remix IDE para codificação e teste dos contratos, além do *Ethereum Wallet* para implantação na rede de teste *Kovan*.

O sistema foi projetado para gerenciar diversos fluxos de trabalho médicos, incluindo:

- Emissão automatizada de prescrições médicas
- Compartilhamento seguro de resultados de exames laboratoriais
- Gestão de interações entre pacientes e profissionais de saúde
- Controle de acesso a procedimentos médicos complexos, como cirurgias e ensaios clínicos

A Figura 3.1 ilustra a arquitetura proposta por (KHATOON, 2020), evidenciando a separação entre os dados mantidos na *blockchain* e os armazenados externamente.

Cada bloco do *Ethereum Node* registra uma lista de controle de acesso (*access control list*) e suas transações, o *hash* da localização dos dados (*hash of data location*) e a identidade do proprietário (*owner identity*), encadeados pela referência ao bloco anterior. Os registros médicos (*EHR*) não são gravados na cadeia, mas sim em um banco de dados local, de forma cifrada (*encrypted data*), enquanto a *blockchain* mantém apenas seus *hashes* e as permissões de acesso. A interação ocorre por meio de uma interface web, que se comunica com a rede via contratos inteligentes, os quais concedem ou solicitam acesso (*provide/request access via smart contract*). Deste modo, o uso dos contratos inteligentes elimina as ineficiências administrativas e reforça a integridade dos dados.

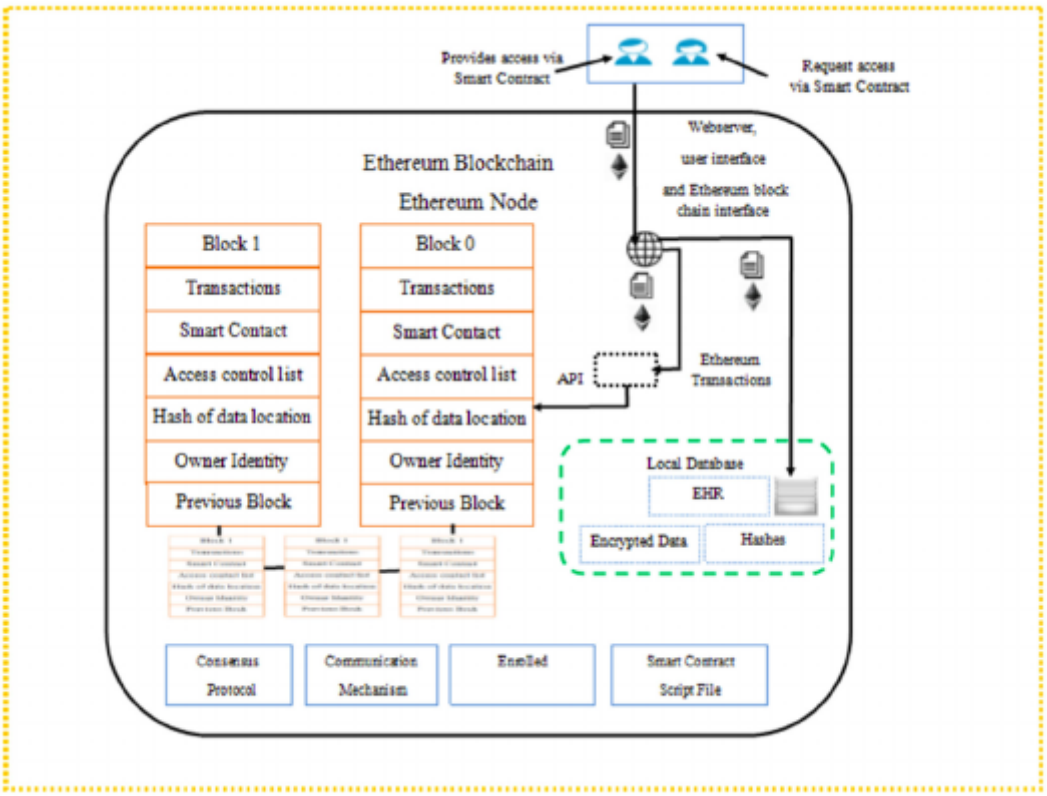


Figura 3.1 – Fluxo de trabalho com acesso controlado por contrato inteligente (KHATOON, 2020).

Um aspecto relevante do estudo é a análise de viabilidade econômica, que demonstrou custos de implantação factíveis para aplicações reais na área da saúde. Esta análise contribui significativamente para a validação prática da tecnologia *blockchain* no contexto médico.

O trabalho de Khatoon estabelece um importante precedente para a aplicação de contratos inteligentes na gestão de fluxos de trabalho médicos. Contudo, o estudo limita-se à implementação em rede de teste, não abordando desafios de escalabilidade em ambiente de produção com alto volume de transações.

3.2 A Consent Model for Blockchain-Based Health Data Sharing Platforms

Este estudo apresentado por (JAIMAN; UROVI, 2020) desenvolve um modelo de consentimento dinâmico baseado em *blockchain* para o controle de consentimento no compartilhamento de dados de saúde, utilizando contratos inteligentes para representar dinamicamente as preferências individuais dos pacientes.

A plataforma implementada utiliza a *blockchain Ethereum* como infraestrutura base, permitindo que contratos inteligentes gerenciem o consentimento individual sobre dados de saúde.

O modelo fundamenta-se em duas ontologias principais:

- ***Data Use Ontology* (DUO):** Modela o consentimento individual dos usuários, definindo as condições específicas para uso dos dados
- ***Automatable Discovery and Access Matrix* (ADA-M):** Descreve as consultas e requisições dos solicitantes de dados

A Figura 3.2 apresenta a arquitetura do modelo de consenso. No fluxo, o *Data Provider* configura seu consentimento por meio da ontologia DUO, enquanto o *Data Requester* descreve suas requisições via ADA-M. Ambos interagem com o contrato inteligente central (*Data Sharing Smart Contract Agreement*) que media o compartilhamento e libera o acesso mediante um *token* de acordo (*Access with Agreement Token*), garantindo que o consentimento individual seja respeitado.

A implementação do modelo foi realizada na *blockchain* da Ethereum, utilizando contratos inteligentes escritos em Solidity. O modelo foi avaliado em diferentes cenários de compartilhamento de dados para garantir sua eficiência e adaptabilidade.

Para avaliar o modelo foi testado em diversos cenários, incluindo diferentes perfis de provedores de dados e de solicitantes desses dados. Os resultados demonstraram flexibilidade adequada para controle de acesso, mantendo o consentimento individual de todos os participantes.

Um aspecto crucial do trabalho é a conformidade com o Regulamento Geral sobre a Proteção de Dados (GDPR) da União Europeia, estabelecendo um precedente importante para a implementação de soluções *blockchain* em conformidade com regulamentações de privacidade internacionais.

Este trabalho contribui significativamente para o campo ao demonstrar como ontologias podem ser integradas com contratos inteligentes para criar sistemas de con-

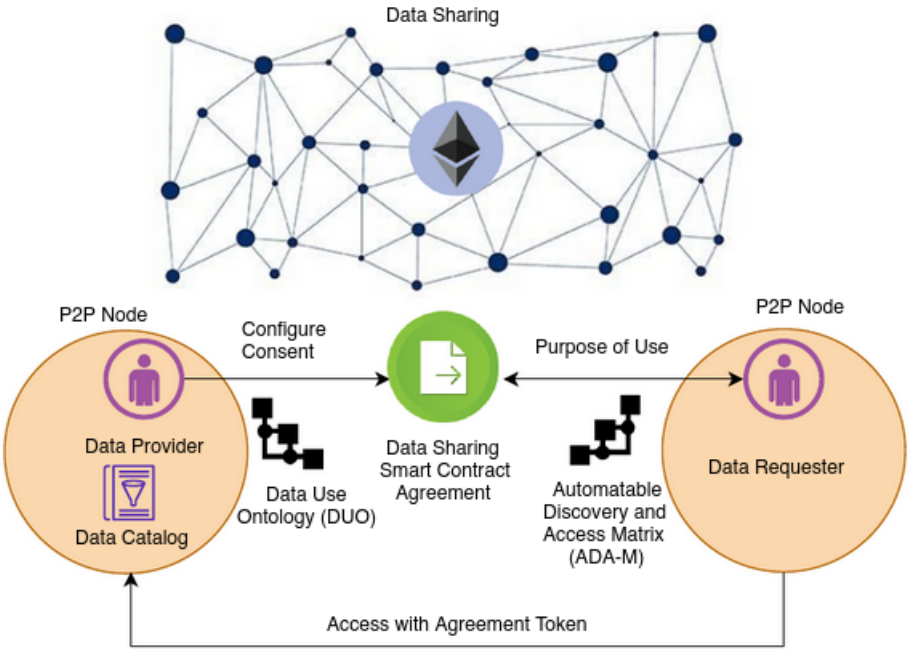


Figura 3.2 – Modelo de consenso baseado na arquitetura de *blockchain* com o uso de contratos inteligentes (JAIMAN; UROVI, 2020).

sentimento dinâmico. A abordagem oferece uma solução técnica para um dos principais desafios regulatórios no compartilhamento de dados de saúde.

3.3 Using Blockchain and Semantic Web Technologies for the Implementation of Smart Contracts Between Individuals and Health Insurance Organizations

Um estudo relevante na área é o trabalho de (CHONDROGIANNIS et al., 2022), no qual foi desenvolvida uma aplicação descentralizada (*DApp*) utilizando a tecnologia *blockchain* e web semântica para permitir a execução e criação de contratos inteligentes entre indivíduos e organizações de seguros de saúde.

Os contratos inteligentes foram escritos em Solidity, e o sistema foi implementado na rede Ethereum. A aplicação foi projetada para facilitar acordos entre indivíduos e organizações de seguros de saúde. A *blockchain* foi utilizada para o armazenamento dos contratos inteligentes, enquanto os dados de saúde dos segurados foram armazenados fora da *blockchain*. O modelo de dados utilizou ontologias e padrões de saúde internacionais para a representação semântica dos dados e dos termos do contrato.

O objetivo dos *smart contracts* foi o gerenciamento de diferentes aspectos dos contratos de seguro de saúde. Foram desenvolvidos vários *smart contracts* para gerenciar esses contratos, como a criação, negociação, assinatura e execução dos termos do contrato. Esses contratos permitiam que indivíduos assinassem contratos de seguro de saúde, pagando a quantia necessária e sendo compensados automaticamente quando os termos do contrato fossem cumpridos.

Este estudo contribui ao demonstrar como a combinação de tecnologias *blockchain* e web semântica pode ser utilizada para melhorar a gestão de contratos de saúde, oferecendo uma solução segura, transparente e eficiente para o gerenciamento e a execução de contratos inteligentes.

3.4 Security and Privacy of Electronic Health Records: Decentralized and Hierarchical Data Sharing Using Smart Contracts

O trabalho de (ZAGHLOUL; LI; REN, 2019) propõe um esquema inovador para o compartilhamento de gestão de dados que empodera os pacientes sobre os seus registros, utilizando os benefícios de segurança e privacidade da tecnologia *blockchain* e dos contratos inteligentes.

O esquema proposto minimiza a dependência das instituições geradoras de registros e permite que pacientes compartilhem seletivamente partes de seus registros com base nas preferências de privacidade desejadas, dando aos pacientes controle sobre como seus dados são compartilhados e com quem. Além disso, impede que membros da equipe combinem suas permissões de acesso para acessar partes dos registros às quais não estão autorizados individualmente. Isso é alcançado através de um processo de verificação inicial realizado por contratos inteligentes.

Utiliza-se criptografia de chave assimétrica para garantir a integridade dos dados; os registros são armazenados fora da *blockchain*, enquanto as políticas de acesso são incorporadas em contratos inteligentes implantados na *blockchain*.

O desempenho do esquema é medido pelo número de computações dos contratos inteligentes e pela sua complexidade. As otimizações nas políticas de acesso são necessárias para melhorar o desempenho geral, reduzindo o custo de execução dos contratos inteligentes.

3.5 Breve Análise Comparativa e Diferenciais do Presente Trabalho

Os trabalhos relacionados estabelecem uma base sólida ao empregar *smart contracts* e automação de fluxos de trabalho, estratégias também adotadas nesta pesquisa. Os estudos de (KHATOON, 2020), (JAIMAN; UROVI, 2020) e (ZAGHLOUL; LI; REN, 2019) concentram-se no controle de acesso e no consentimento por meio dos contratos inteligentes, enquanto (CHONDROGIANNIS et al., 2022) avança na representação semântica dos termos contratuais com o uso de ontologias. No armazenamento de dados sensíveis, (JAIMAN; UROVI, 2020), (ZAGHLOUL; LI; REN, 2019) e (CHONDROGIANNIS et al., 2022) convergem ao mantê-los fora da *blockchain* (*off-chain*), preservando na cadeia apenas as políticas de acesso ou os termos de contrato.

É nesse ponto que o presente trabalho se diferencia. A BlockHealth avança ao empregar o *Interplanetary File System* (IPFS) para o armazenamento distribuído dos prontuários e, principalmente, ao adicionar uma camada de segurança sobre esse armazenamento. Reconhecendo que o IPFS não oferece criptografia de conteúdo de forma nativa (IPFS Docs, n.d.b), os arquivos médicos são cifrados antes do envio à rede. Desta forma os dados permanecem protegidos, diferentemente das abordagens anteriores, que armazenam em repositórios externos sem detalhar essa proteção.

Um segundo diferencial reside na avaliação de custos. Esta pesquisa tende a examinar o custo computacional das operações, quantificando o *gas* consumido e projetando seu impacto financeiro na rede principal Ethereum (Seção 6.6). Essa análise fornece uma visão mais concreta sobre a viabilidade da solução em ambiente de produção.

Em síntese, a contribuição da BlockHealth propõe um modelo tecnicamente viável e mais seguro para o armazenamento efetivo de dados de saúde, não se limitando somente ao acesso via *smart contracts*, acompanhado de uma avaliação de custo ausente nos trabalhos correlatos.

4. METODOLOGIA

Este capítulo apresenta a metodologia que foi adotada para o desenvolvimento de uma solução baseada na tecnologia *blockchain* para a gestão segura de dados médicos. São descritos os procedimentos metodológicos, a caracterização da pesquisa, os critérios de seleção tecnológica e as etapas de desenvolvimento e validação da solução proposta.

A escolha desta abordagem metodológica justifica-se pela necessidade de desenvolver uma solução prática para problemas reais identificados na gestão de dados médicos, bem como pela necessidade de validar empiricamente a eficácia da tecnologia *blockchain* neste contexto específico.

4.1 Definição do Problema e Objetivos

A gestão médica enfrenta desafios significativos relacionados à segurança, privacidade e eficiência no compartilhamento de informações entre diferentes atores do sistema de saúde. Entre eles, destacam-se vulnerabilidades de segurança em sistemas centralizados, a falta de controle granular de acesso, a ausência de mecanismos de auditoria transparentes e ineficiências em processos burocráticos.

Diante deste cenário, este trabalho teve como objetivo o desenvolvimento de uma aplicação descentralizada baseada em *blockchain* para a gestão de dados médicos, capaz de oferecer controle granular de acesso, rastreabilidade das operações e automatização de processos por meio de contratos inteligentes, demonstrando sua viabilidade técnica em ambiente controlado.

4.2 Seleção da Plataforma Blockchain

A seleção da plataforma *blockchain* constitui um elemento fundamental para o sucesso da implementação. Esta escolha foi baseada em critérios técnicos objetivos e mensuráveis, permitindo uma análise comparativa sistemática entre as principais alternativas disponíveis.

Os critérios estabelecidos para a seleção da plataforma *blockchain* incluem:

- **Segurança:** Robustez dos mecanismos de consenso, histórico de vulnerabilidades e recursos nativos de proteção de dados
- **Escalabilidade:** Capacidade de processamento de transações (TPS), tempo de confirmação de blocos e eficiência energética

- **Viabilidade Econômica:** Custos de transação, complexidade de implementação e manutenção da infraestrutura
- **Suporte a Contratos Inteligentes:** Maturidade da linguagem de programação, ferramentas de desenvolvimento disponíveis e qualidade da documentação
- **Comunidade e Ecossistema:** Tamanho da comunidade de desenvolvedores, disponibilidade de bibliotecas e *frameworks*

Com base nesses critérios, optou-se pela plataforma Ethereum. Sua escolha justifica-se pela maturidade do ecossistema de contratos inteligentes, pela ampla comunidade de desenvolvedores e pela robustez de suas ferramentas e documentação. Além de disponibilizar a linguagem Solidity, específica para o desenvolvimento de contratos inteligentes, e de um conjunto consolidado de ferramentas de desenvolvimento e teste, como Truffle e Ganache, que viabilizaram a implementação e a validação da solução em ambiente local.

4.3 Desenvolvimento dos Contratos Inteligentes

Os contratos inteligentes ou *smart contracts* constituem o núcleo funcional da solução proposta, implementando a lógica de negócios e os mecanismos de controle de acesso de forma descentralizada e automatizada. Para o seu desenvolvimento, utilizou-se a linguagem Solidity, oficial e especificamente projetada para a plataforma Ethereum. Esta escolha justifica-se pela extensa documentação da linguagem, sua maturidade e o amplo suporte da comunidade de desenvolvedores.

A implementação contemplou a automatização da emissão de receitas, assegurando que apenas profissionais autorizados possam emitir estas prescrições. O foco principal, contudo, foi o compartilhamento seguro de dados sensíveis, incluindo dados laboratoriais e resultados de exames entre médicos e pacientes, com um controle de acesso rigoroso.

Outro quesito de avaliação é a interação dos pacientes com os profissionais de saúde. Para isso, utilizam-se contratos inteligentes no gerenciamento e registro de transações e interações na *blockchain*. Esses contratos garantiram a transparência, a segurança e a integridade dos dados trocados, proporcionando maior confiança entre todas as partes envolvidas.

4.4 Aspectos de Segurança e Privacidade

A implementação de medidas robustas de segurança e privacidade é fundamental para garantir a confiabilidade da solução no contexto sensível da gestão de dados médicos.

Nesse contexto, para garantir a segurança, o sistema foi desenvolvido com base nos seguintes princípios de segurança:

- **Imutabilidade:** Garantia de que todas as transações registradas na *blockchain* sejam protegidas contra alterações não autorizadas através de mecanismos criptográficos
- **Controle de Acesso:** Implementação de políticas de acesso baseadas em papéis (RBAC) nos contratos inteligentes, definindo permissões específicas para cada tipo de usuário
- **Criptografia:** Utilização de algoritmos de criptografia simétrica (AES-256-GCM) para proteção dos dados sensíveis antes do armazenamento, com derivação segura de chaves
- **Auditoria:** Registro detalhado de todas as operações realizadas no sistema, permitindo rastreabilidade completa das ações

4.5 Desenvolvimento da Aplicação

O ambiente de desenvolvimento foi estruturado para garantir a qualidade e a eficiência na implementação da solução. A prototipagem inicial e os testes rápidos foram conduzidos no Remix IDE, enquanto o desenvolvimento mais avançado utilizou o Visual Studio Code com extensões para Solidity. Para gestão do projeto, compilação, testes e *deployment* dos contratos inteligentes, empregou-se o *Truffle Suite*[1], e o Ganache forneceu uma *blockchain* local para o desenvolvimento e testes iniciais.

O desenvolvimento seguiu um ciclo iterativo. Para cada alteração nos contratos inteligentes, estes eram recompilados com o `truffle compile`, e reimplantados na *blockchain* local por meio dos *scripts* de migração através do `truffle migrate`, responsáveis por estabelecer as referências de endereço entre os três contratos do sistema. Em seguida eram executadas as suítes de testes automatizados, com o `truffle test`, integrado aos *frameworks* Mocha e Chai, validando cada funcionalidade antes do prosseguimento.

[1] Em setembro de 2023, a Consensys anunciou a descontinuação do Truffle Suite e do Ganache, recomendando a migração para ferramentas como Hardhat e Foundry. As ferramentas, contudo, permaneceram funcionais durante o desenvolvimento deste trabalho.

Este processo de compilação, implantação e teste foi repetido sucessivamente ao longo da implementação, assegurando a verificação isolada antes da introdução de novas funcionalidades. A camada de API foi desenvolvida em Node.js com o *framework* Express.js, e a integração com o armazenamento distribuído utilizou um nó IPFS instanciado por meio da biblioteca Helia. Os resultados consolidados dessa etapa de validação são apresentados no Capítulo 6.

4.6 Uso de Ferramentas de Inteligência Artificial

Cabe registrar que as ferramentas de inteligência artificial Claude Opus 4.8 (Anthropic) e Gemini Pro (Google), acessadas em 2026, foram utilizadas como apoio à revisão textual e à organização da escrita, atuando exclusivamente sobre o texto redigido pelo autor. Todas as sugestões foram revisadas criticamente, sendo o conteúdo de responsabilidade integral do autor.

5. BLOCKHEALTH

Este capítulo apresenta a aplicação descentralizada *BlockHealth*, detalhando sua arquitetura, implementação e aspectos técnicos. A proposta é oferecer uma solução tecnológica para os problemas de gestão médica apresentados anteriormente, utilizando a *blockchain* e contratos inteligentes para garantir segurança, privacidade e descentralização no compartilhamento de dados de saúde.

Como visto nos capítulos anteriores, os desafios da gestão de dados médicos incluem questões críticas de segurança, privacidade, eficiência e interoperabilidade. A aplicação *BlockHealth* aborda essas problemáticas por meio da implementação de contratos inteligentes na *blockchain*, proporcionando uma solução descentralizada e segura para o compartilhamento de informações médicas.

Todo o código do projeto está disponível publicamente no *Github* através do link: https://github.com/riannbarbosa/BlockHealth.

5.1 Visão Geral da Aplicação

A aplicação *BlockHealth* foi desenvolvida como uma solução descentralizada para o compartilhamento seguro de dados médicos, tendo como principal tecnologia a *blockchain* Ethereum e contratos inteligentes. O sistema foi projetado para atender às necessidades específicas de diferentes atores do ecossistema de saúde, proporcionando controle granular de acesso e garantindo a integridade dos dados compartilhados.

5.1.1 Estrutura

A implementação da estrutura de arquivos da API se organiza em pastas específicas definidas a seguir:

- **contracts**/: Contém os contratos inteligentes Solidity que definem a lógica de negócio da *blockchain*.
  - `AdminContract.sol`: Gerencia funcionalidades administrativas do sistema.
  - `MedicContract.sol`: Define operações relacionadas aos profissionais médicos.
  - `PatientContract.sol`: Implementa a lógica de gerenciamento de pacientes.
- **migrations**/: Pasta que contém os scripts de implantação dos contratos no ambiente Ganache.

  - `1_MedicContract_migration.js`: Script de deploy do contrato médico.
  - `2_AdminContract_migration.js`: Script de deploy do contrato administrativo.
  - `3_PatientContract_migration.js`: Script de deploy do contrato de pacientes.

- **test**/: Contém os testes nos contratos e de integração dos contratos inteligentes.
  - `AdminContract.test.js`: Testes das funcionalidades administrativas.
  - `MedicContract.test.js`: Testes das operações médicas.
  - `PatientContract.test.js`: Testes do gerenciamento de pacientes.

- **src**/: Diretório principal contendo o código-fonte da API REST.
  - **controllers**/: Implementa a lógica de negócio que conecta as requisições HTTP aos contratos *blockchain*.
    - `adminController.js`: Controla funcionalidades administrativas.
    - `doctorController.js`: Controla e gerencia operações relacionadas a médicos.
    - `patientController.js`: Controla funcionalidades de pacientes.
  - `ipfs/`: Gerencia o armazenamento descentralizado e configura a criptografia dos arquivos.
    - `ipfs.js`: Configura a conexão com a rede IPFS.
    - `file.js`: Implementa funções de upload e download de arquivos.
  - `routes/`: Define os *endpoints* para cada entidade do sistema.
    - `adminRoutes.js`: Rotas para operações administrativas.
    - `doctorRoutes.js`: Rotas para funcionalidades médicas.
    - `patientRoutes.js`: Rotas para gerenciamento de pacientes.
  - **utils**/: Contém utilitários de configuração e inicialização.
    - `contractUtils.js`: Inicializa e gerencia instâncias dos contratos.
    - `test_server.js`: Configurações para ambiente de testes.
  - `app.js`: Centraliza a configuração das rotas da API e inicializa a documentação *Swagger*.
  - `index.js`: Ponto de entrada da aplicação que inicializa o servidor *Express*.
  - **uploads**/: Diretório temporário para armazenamento de arquivos antes do envio ao IPFS.

5.1.2 Stakeholders do Sistema

O sistema utiliza uma abordagem hierárquica de controle de acesso, onde os níveis de permissão são definidos para três tipos principais de usuários:

- **Administrador**: Responsável pelo fluxo de administração geral do sistema. Este nível de acesso permite adicionar e remover usuários e entidades médicas por meio da API administrativa. A API administrativa conecta-se ao *smart contract* associado que opera na *blockchain*.
- **Profissional de Saúde**: Médicos, enfermeiros e outros profissionais autorizados que podem enviar registros médicos/prontuários e realizar o download de dados médicos de pacientes aos quais tem acesso autorizado. Essas operações são realizadas através da API médica, que posteriormente se conecta ao *smart contract* e à *blockchain*.
- **Paciente**: Proprietário dos dados médicos, com capacidade de enviar e realizar o download de seus próprios registros, utilizando a API do paciente. Esta API, por sua vez, conecta-se à rede *blockchain* em conjunto com o *smart contract* correspondente.

5.2 Arquitetura do sistema

A Figura 5.1 apresenta a estrutura geral da aplicação *BlockHealth*, seguindo uma arquitetura de três camadas, para garantir a segurança dos dados médicos e facilitar a manutenção. A arquitetura é composta por camada de API REST, camada de contratos inteligentes e o armazenamento descentralizado (IPFS).

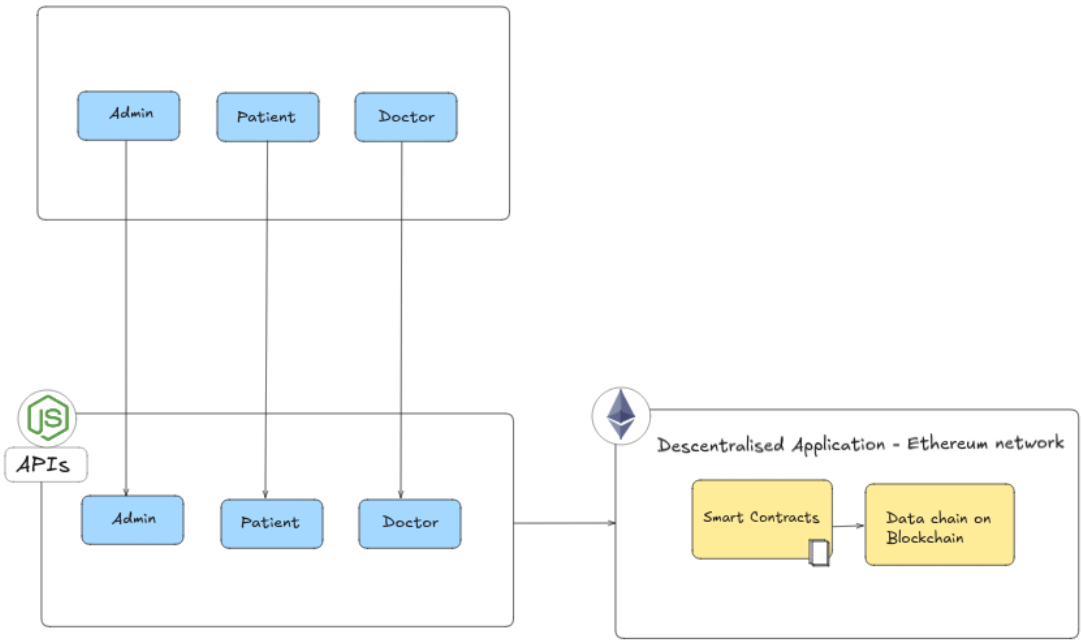


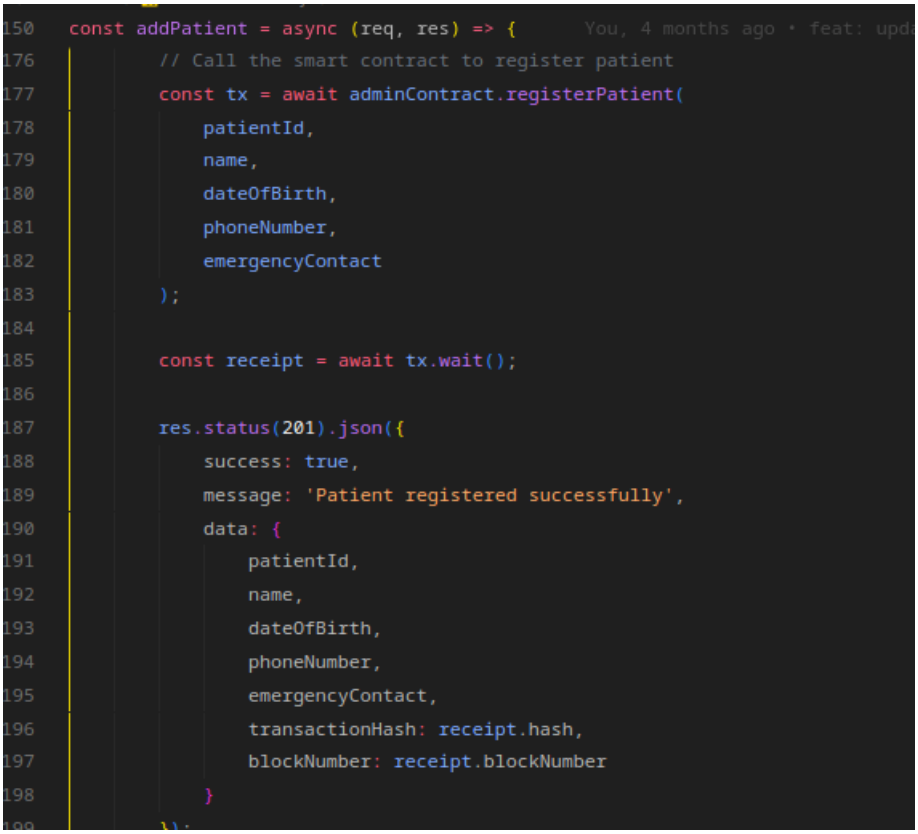
Figura 5.1 – Arquitetura da aplicação. Fonte: Elaboração Própria

5.2.1 Camada do Backend (API)

A camada desenvolvida em *Node.js* com o *framework Express.js* atua como um intermediário entre as interfaces do usuário e a *blockchain*. Esta camada é responsável por validar dados antes do envio à *blockchain* e fornecer interface RESTful para aplicações cliente.

A API é estruturada em quatro módulos centrais, cada um projetado para atender um conjunto específico de funcionalidades: o módulo administrativo, o módulo médico, o módulo de pacientes e o módulo de gerenciamento de arquivos.

Na parte administrativa, a API é responsável por gerenciar operações administrativas como cadastro, revogação de médicos através do *AdminContract*, registro e desativação de pacientes. Também implementa validações de endereços Ethereum e controle de acesso através do contrato inteligente. Seus principais *endpoints* incluem `POST /api/admin/patients` para registro de pacientes e `POST /api/admin/doctors` para registro médicos.



```
150  const addPatient = async (req, res) => {      You, 4 months ago • feat: upda
176      // Call the smart contract to register patient
177      const tx = await adminContract.registerPatient(
178          patientId,
179          name,
180          dateOfBirth,
181          phoneNumber,
182          emergencyContact
183      );
184
185      const receipt = await tx.wait();
186
187      res.status(201).json({
188          success: true,
189          message: 'Patient registered successfully',
190          data: {
191              patientId,
192              name,
193              dateOfBirth,
194              phoneNumber,
195              emergencyContact,
196              transactionHash: receipt.hash,
197              blockNumber: receipt.blockNumber
198          }
```

Figura 5.2 – Função de registro de um paciente. Fonte: Elaboração Própria

A Figura 5.2 mostra a função `addPatient`, chamada no `adminController.js`, que é um `POST /api/admin/patients` para o registro de um paciente. Esta função, por sua vez, chama a função `registerPatient` do contrato inteligente, enviando os dados como parâmetro.

Na parte médica, os médicos autorizados podem adicionar registros médicos ao prontuário dos pacientes, realizando o *upload* de arquivos para a IPFS. Sua principal operação inclui o *endpoint* `POST /api/doctor/records` de inclusão destes registros.

```
const addPatientRecord = async (req, res) => {

        let cid = '';
        let fileName = '';
        if (medicalFile) {
            try {
                const ipfsResult = await uploadToIPFS(medicalFile.path, patientId, true);
                cid = ipfsResult.cid;
                fileName = medicalFile.originalname;

                fs.unlinkSync(medicalFile.path);
            } catch (ipfsError) {
                console.error('IPFS upload error:', ipfsError);
                cid = 'QmNoPhuLRhyaJzx1KoJ6vgmwmvFTxc8a1Cv7Bx1RPb6cNq'; // Placeholder
                fileName = medicalFile ? medicalFile.originalname : 'No file';
            }
        } else {
            cid = 'QmTextOnlyRecord' + Date.now();
            fileName = 'Text record only';
        }

        // Add medical record to blockchain
        const tx = await medicContract.addMedicalRecord(
            cid,
            fileName,
            patientId,
            diagnosis,
            treatment
        );
```

Figura 5.3 – Função de registro de arquivo médico. Fonte: Elaboração Própria

A função `addPatientRecord`, apresentada na Figura 5.3, processa o registro de prontuários médicos em duas etapas. Primeiramente, ao receber o arquivo médico, a função o encaminha para o método `uploadToIPFS` que realiza o armazenamento no IPFS e retorna um *content-identifier* (CID) único. Posteriormente, este CID é registrado na *blockchain* através da invocação da função `addMedicalRecord` do contrato inteligente, estabelecendo um vínculo imutável entre o arquivo armazenado e o registro médico do paciente.

O módulo de pacientes fornece acesso aos próprios registros, tanto os adicionados por médicos quanto os auto-cadastrados, permitindo gerenciamento completo de seu histórico médico. Os principais *endpoints* incluem `GET /api/patient/:patientId/medical-records` para consultar registros criados por profissionais de saúde e o `POST /api/patient/upload-record` para adicionar novos arquivos pessoais ao sistema.

Por fim, há um módulo adicional para arquivos que gerencia operações de *upload* e *download* via IPFS, incluindo criptografia AES-256-GCM para arquivos sensíveis. O *endpoint* `GET /api/files/download/:cid` permite o *download* de arquivos armazenados

no IPFS. A Figura 5.4 descreve o fluxo de recuperação de dados. O processo inicia-se com a requisição dos *chunks* do arquivo armazenado na rede IPFS, utilizando o módulo `fsUnix` da biblioteca Helia. Em seguida, é verificado se o conteúdo requer criptografia. Caso necessário, uma chave simétrica é derivada com base no identificador do paciente por meio do método `generateEncryptionKey`, a fim de possibilitar a decodificação dos dados antes de seu retorno.

```
const downloadFromIPFS = async (cid, patientId = null, isEncrypted = false) => {      You, 5
  try {
    const { fsUnix } = await getHelia();

    // Download file from IPFS
    const chunks = [];
    for await (const chunk of fsUnix.cat(cid)) {
      chunks.push(chunk);
    }
    let fileBuffer = Buffer.concat(chunks);

    // Decrypt if necessary
    if (isEncrypted && patientId) {
      console.log(`Decrypting file for patient ${patientId}`);
      const encryptionKey = generateEncryptionKey(patientId);
      fileBuffer = decryptContent(fileBuffer, encryptionKey);
      console.log('File decrypted successfully');
    }

    return fileBuffer;
```

Figura 5.4 – Função para download IPFS. Fonte: Elaboração Própria

5.2.1.1 *Swagger*

Por se tratar de uma API e visando disponibilizar uma interface visual de interação e consulta, foi utilizada a ferramenta *Swagger*. O *Swagger* é uma solução *open-source* amplamente adotada para documentação de APIs, permitindo explorar os recursos de forma dinâmica e interativa. A Figura 5.5 abaixo mostra a aplicação ativa no *Swagger* no *endpoint* `/api-docs`.

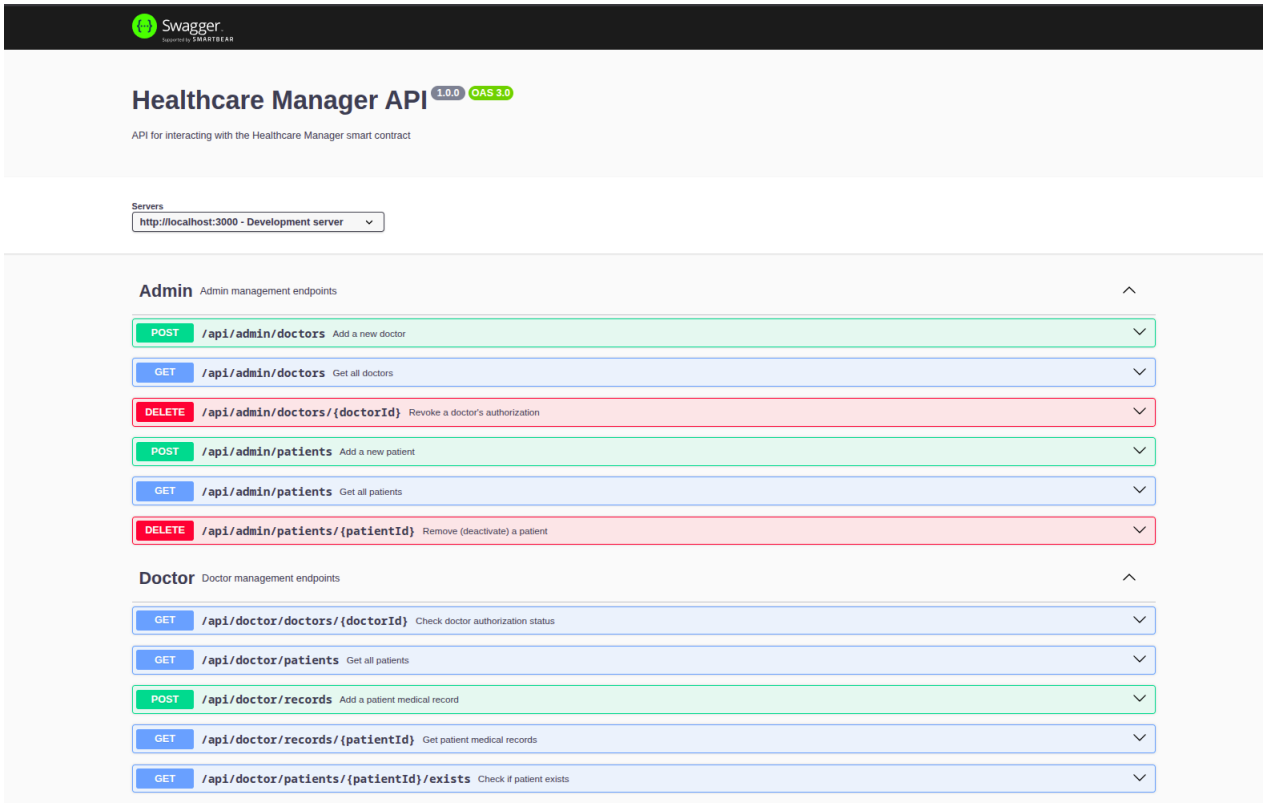
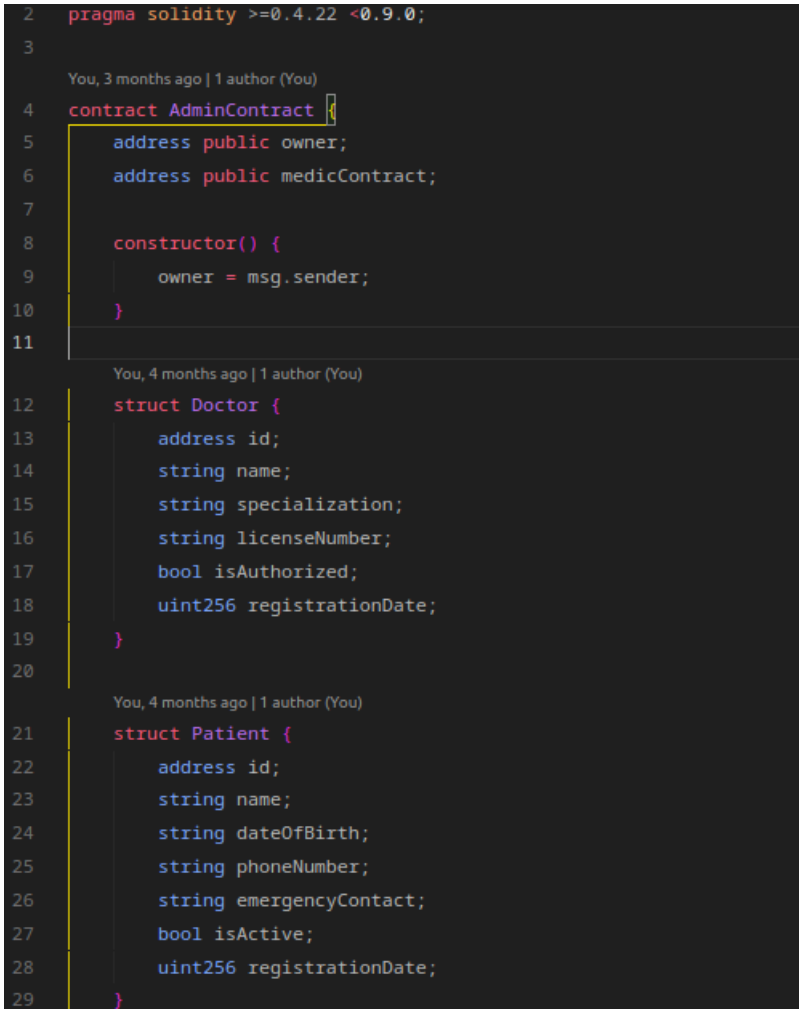


Figura 5.5 – Documentação *Swagger*. Fonte: Elaboração Própria

5.2.2 Camada de *Blockchain*

Implementada em Solidity, esta camada constitui o núcleo da aplicação e garante a imutabilidade. É onde constam os contratos inteligentes, três principais contratos interconectados que se comunicam através de referências de endereços e chamadas inter-contratos.

O *AdminContract* é responsável pelo gerenciamento de todas as entidades do sistema (médicos e pacientes). A Figura 5.6 apresenta a estrutura de dados do contrato, onde há um `struct Doctor` para o armazenamento dos dados médicos e um `struct Patient` para os dados dos pacientes.



```
2   pragma solidity >=0.4.22 <0.9.0;
3
    You, 3 months ago | 1 author (You)
4   contract AdminContract {
5       address public owner;
6       address public medicContract;
7
8       constructor() {
9           owner = msg.sender;
10      }
11
        You, 4 months ago | 1 author (You)
12      struct Doctor {
13          address id;
14          string name;
15          string specialization;
16          string licenseNumber;
17          bool isAuthorized;
18          uint256 registrationDate;
19      }
20
        You, 4 months ago | 1 author (You)
21      struct Patient {
22          address id;
23          string name;
24          string dateOfBirth;
25          string phoneNumber;
26          string emergencyContact;
27          bool isActive;
28          uint256 registrationDate;
29      }
```

Figura 5.6 – Estrutura dos dados. Fonte: Elaboração Própria

O contrato também implementa o *modifier* `onlyOwner` (linha 9), que restringe as operações administrativas apenas ao proprietário do contrato. Esse mecanismo garante que apenas administradores autorizados possam revogar e registrar entidades no sistema.

O *MedicContract* armazena e gerencia os registros médicos criados por profissionais de saúde autorizados. Sua estrutura de dados inicial é apresentada na Figura 5.7, que exibe as variáveis de referência aos demais contratos e o `struct MedicalRecord`. Além disso, o contrato implementa o *modifier* `onlyAuthorizedDoctor` como controle de acesso, validando se um médico está autorizado por meio da consulta ao *AdminContract*.

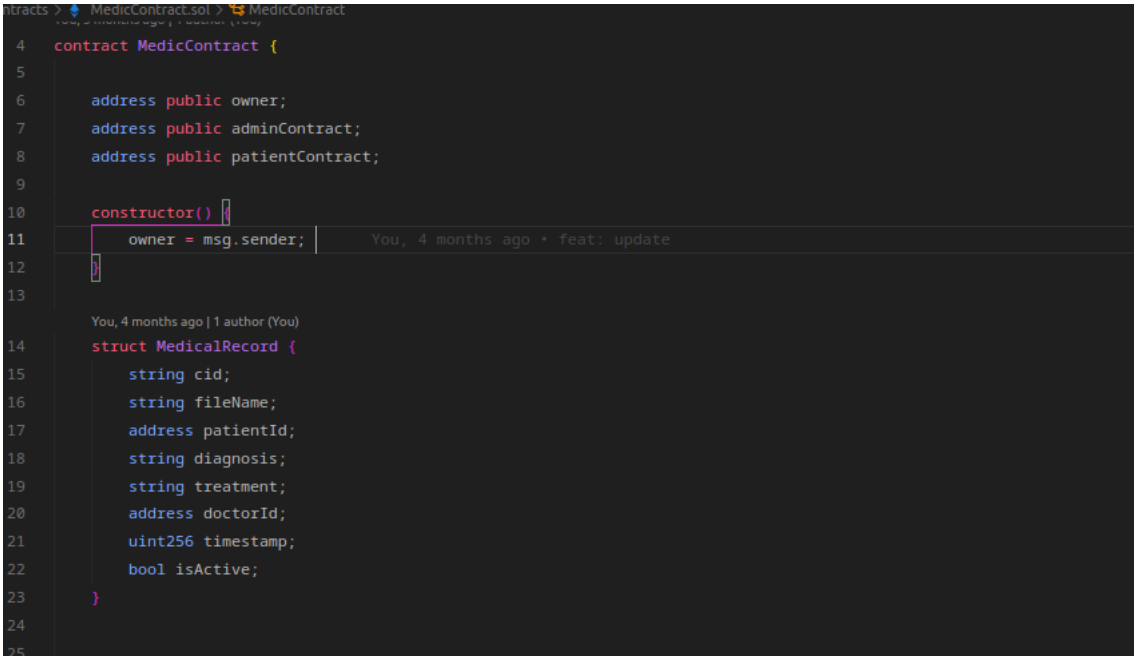
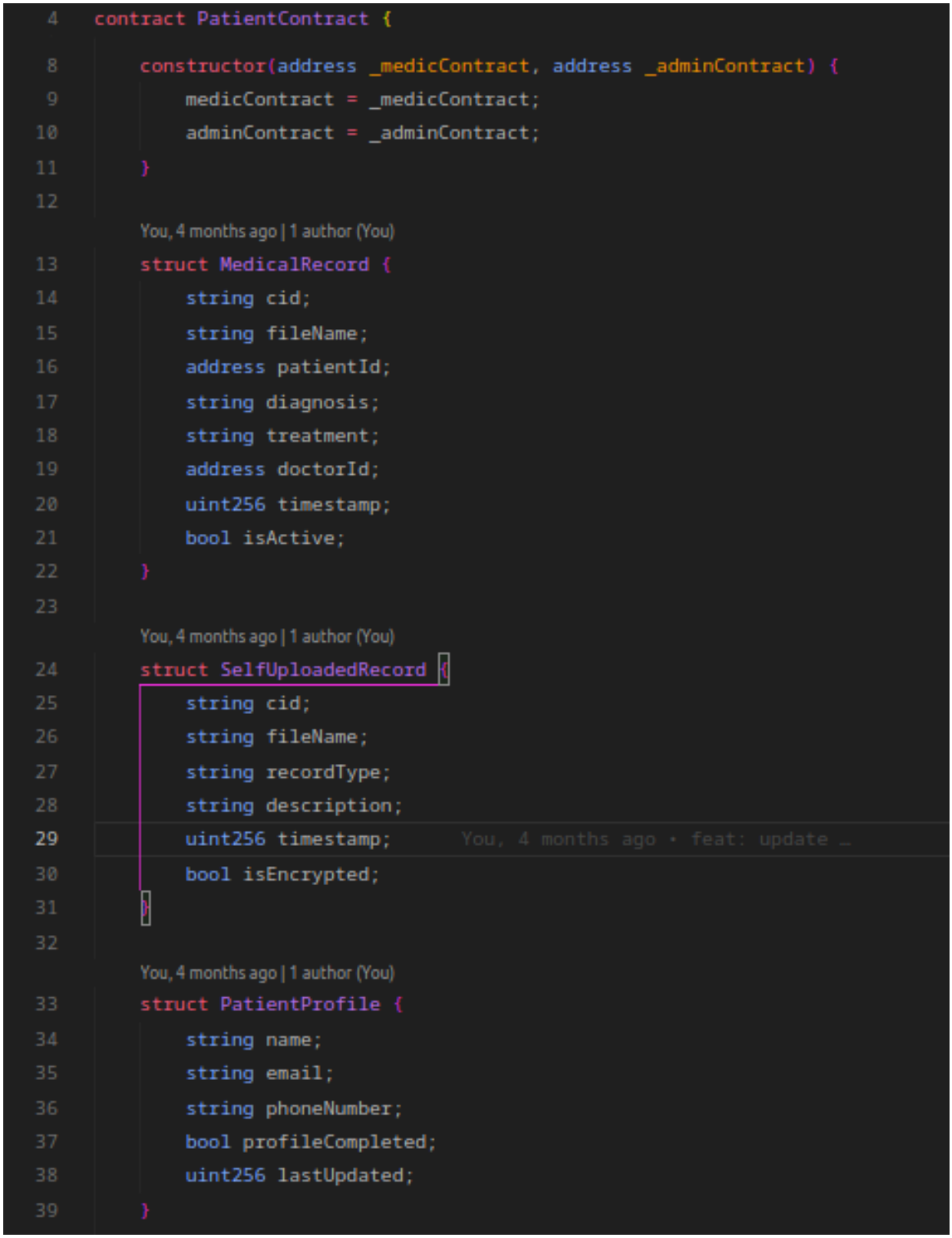


Figura 5.7 – Contrato medico. Fonte: Elaboração Própria

Para o *PatientContract* foram implementado integrações, com o contrato mantendo referências ao *MedicContract* e *AdminContract* que são chamada no `constructor`. A estrutura contém: o `struct SelfUploadedRecord` para armazenar dados de documentos junto com o CID do IPFS e o `struct PatientProfile` para armazenar os dados de perfil do paciente. Além de ter controles de acesso como `onlyPatient` para garantir que apenas pacientes registrados possam operar, `onlyValidPatient` para validar a existência do paciente e `onlyPatientOrDoctor` para verificar o acesso tanto a pacientes quanto a médicos autorizados. A Figura 5.8 abaixo mostra a estrutura inicial do contrato.



```
4   contract PatientContract {

8       constructor(address _medicContract, address _adminContract) {
9           medicContract = _medicContract;
10          adminContract = _adminContract;
11      }
12

        You, 4 months ago | 1 author (You)
13      struct MedicalRecord {
14          string cid;
15          string fileName;
16          address patientId;
17          string diagnosis;
18          string treatment;
19          address doctorId;
20          uint256 timestamp;
21          bool isActive;
22      }
23

        You, 4 months ago | 1 author (You)
24      struct SelfUploadedRecord {
25          string cid;
26          string fileName;
27          string recordType;
28          string description;
29          uint256 timestamp;      You, 4 months ago • feat: update …
30          bool isEncrypted;
31      }
32

        You, 4 months ago | 1 author (You)
33      struct PatientProfile {
34          string name;
35          string email;
36          string phoneNumber;
37          bool profileCompleted;
38          uint256 lastUpdated;
39      }
```

Figura 5.8 – Contrato de paciente. Fonte: Elaboração Própria

5.2.2.1 Eventos e Funções

Além das `structs` nos contratos inteligentes, para realizar a inserção dos dados na *blockchain*, há funções de cadastro, remoção de autorização e desativação de paciente,

entre outras. A Figura 5.9 abaixo demonstra um exemplo de cadastro de paciente no `AdminContract`.

```
114    function registerPatient(
115        address _patientId,
116        string memory _name,
117        string memory _dateOfBirth,
118        string memory _phoneNumber,
119        string memory _emergencyContact
120    ) public onlyOwner {
121        require(!patients[_patientId].isActive, "Patient already registered");
122        require(_patientId != address(0), "Invalid patient address");
123
124        patients[_patientId] = Patient({
125            id: _patientId,
126            name: _name,
127            dateOfBirth: _dateOfBirth,
128            phoneNumber: _phoneNumber,
129            emergencyContact: _emergencyContact,
130            isActive: true,
131            registrationDate: block.timestamp
132        });
133
134        patientList.push(_patientId);
135
136        emit PatientRegistered(_patientId, _name);
137    }
```

Figura 5.9 – Função de cadastro. Fonte: Elaboração Própria

Outro ponto é que funções como o cadastro de pacientes emitem eventos para registrar operações realizadas no contrato. Os eventos são declarados por meio da sintaxe `event NomeDoEvento(parâmetros)` e emitidos utilizando `emit`. Esses eventos permitem que aplicações externas monitorem as alterações de estado da *blockchain*, facilitando o rastreamento das operações realizadas.

5.2.3 Camada de Armazenamento Descentralizado (IPFS)

Para evitar o armazenamento de volumes de dados direto na *blockchain*, utiliza-se o *InterPlanetary File System* ou IPFS, um protocolo de rede descentralizada e distribuída que opera em arquitetura de ponto a ponto (*P2P*). Este sistema permite o armazenamento, compartilhamento e acesso seguro a arquivos e dados de forma distribuída, eliminando a dependência de servidores centralizados.

5.2.3.1 Helia

Para este projeto, é utilizada a aplicação Helia, uma implementação do protocolo IPFS que pode ser executada em diferentes ambientes (IPFS / HELIA CONTRIBUTORS, 2025). A finalidade é possibilitar o armazenamento descentralizado de arquivos médicos. Ela é usada para realizar a comunicação direta com a rede IPFS, permitindo instanciar um nó leve dentro do ambiente *Node.js*.

A biblioteca gerencia o sistema de arquivos através da interface `UnixFS`, sendo responsável por adicionar os *buffers* dos arquivos criptografados à rede e recuperá-los posteriormente através de seus respectivos CIDs, garantindo uma integração nativa e eficiente com o *backend*.

5.2.3.2 Endereçamento por Conteúdo

O IPFS implementa um sistema de endereçamento baseado em conteúdo, gerando identificadores únicos denominados *IPFS Content Identifier* ou CID. Cada CID é derivado de um *hash* criptográfico calculado diretamente a partir do conteúdo do arquivo. Isso significa que o conteúdo gera sempre o mesmo CID, fazendo com que qualquer alteração no arquivo gere um CID completamente diferente, e o próprio CID pode ser usado para verificar se o arquivo baixado corresponde ao original.

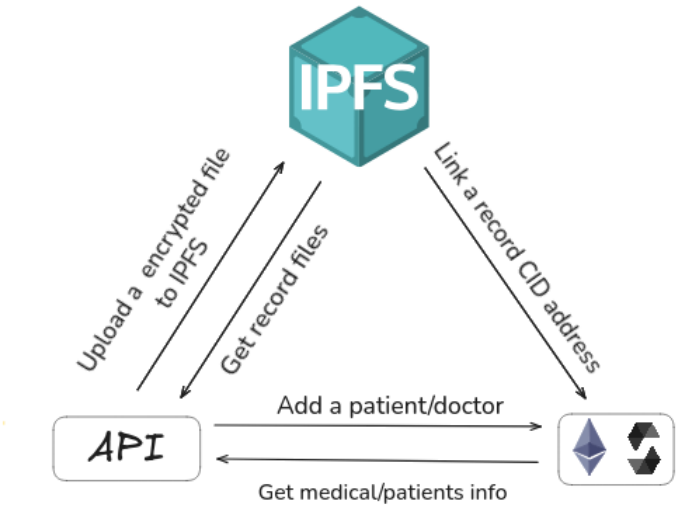


Figura 5.10 – Sistema IPFS. Fonte: Elaboração Própria

A Figura 5.10 ilustra a arquitetura de interação entre a API desenvolvida, a *blockchain* e *smart contracts* e a rede IPFS. Nessa arquitetura, os dados sensíveis nunca trafegam nem são armazenados diretamente na *blockchain*. Inicialmente, a API recebe o arquivo médico e o criptografa de forma local. O conteúdo cifrado é enviado à rede

IPFS, que devolve o CID correspondente. Por fim, este CID, que é um identificador e que funciona como um ponteiro para o dado, é registrado na *blockchain* por meio do *smart contract*, em conjunto com as permissões de acesso. Portanto, o armazenamento de arquivos cabe à rede IPFS, em uma camada *off-chain*.

O envio do arquivo criptografado para IPFS é realizado pela função `uploadToIPFS`, descrita na Figura 5.11. Esta função recebe como parâmetros o caminho do arquivo (`filePath`), o identificador do paciente (`patientId`) e um booleano (`encryptFile`) que indica se o arquivo deve ser criptografado antes do envio.

```
144  const uploadToIPFS = async (filePath, patientId = null, encryptFile = true) => {
145    try {
146      const fileName = path.basename(filePath)
147      let fileBuffer = fs.readFileSync(filePath)
148
149      // Encrypt content if encryption is enabled and patientId is provided
150      if (encryptFile && patientId) {
151        const encryptionKey = generateEncryptionKey(patientId)
152        const encryptionResult = encryptContent(fileBuffer, encryptionKey)
153        fileBuffer = encryptionResult.encryptedData
154      }
155
156      const { fsUnix } = await getHelia()
157      const cid = await fsUnix.addBytes(fileBuffer)
158
159      console.log(`File uploaded to Helia/IPFS: ${fileName} -> ${cid.toString()}`)
160
161      return {
162        cid: cid.toString(),
163        encrypted: !!(encryptFile && patientId),
164      }
165    } catch (error) {
166      console.error('Error uploading to Helia/IPFS:', error)
167      // Fallback to mock
168      const fileName = path.basename(filePath)
169      const mockCID = `Qm${Buffer.from(fileName + Date.now())
170        .toString('hex')
171        .substring(0, 44)}`
172      console.log(`Fallback mock IPFS upload: ${fileName} -> ${mockCID}`)
173
174      return {
175        cid: mockCID,
176        encrypted: false,
177        encryptionMetadata: null,
178      }
179    }
180  }
```

Figura 5.11 – Upload para IPFS. Fonte: Elaboração Própria

Inicialmente, a função lê o conteúdo do arquivo a partir do caminho fornecido. Caso a criptografia esteja habilitada e um identificador de paciente tenha sido informado, gera-se uma chave específica para o paciente por meio da função `generateEncryptionKey`

e o conteúdo é cifrado com a função `encryptContent`, substituindo-se o *buffer* original pelo conteúdo criptografado. Em seguida, o conteúdo é inserido na rede por meio da interface `fsUnix`, obtida a partir de `getHelia`, cujo método `addBytes` retorna o CID correspondente ao dado armazenado.

A função retorna o objeto contendo o CID gerado e o booleano `encrypted`, que registra se o conteúdo foi efetivamente criptografado. Caso a inserção na rede IPFS falhe, é capturada a exceção e é gerado um CID fictício (*mock*) derivado do nome do arquivo, de modo a preservar a continuidade da execução em ambiente de desenvolvimento.

5.2.3.3 Criptografia no IPFS

O processo de armazenamento implementa uma camada adicional de segurança através da criptografia prévia dos dados antes do envio à rede distribuída. Esta abordagem se faz necessária porque o protocolo IPFS oferece apenas criptografia de transporte (*transport-encryption*), que protege a comunicação entre nós da rede, mas não oferece nativamente a criptografia de conteúdo (*content-encryption*) para os dados armazenados (IPFS Docs, n.d.b).

O código apresentado na Listagem 5.1 implementa a geração da chave. A função `generateEncryptionKey` produz uma chave determinística a partir de um *hash* SHA-256, calculado sobre a concatenação de um *secret* predefinido com o *patientId* específico.

```
1 const generateEncryptionKey = patientId => {
2   const secret = process.env.ENCRYPTION_SECRET
3   if (!secret) throw new Error('ENCRYPTION_SECRET nao definido no
      ambiente')
4   return crypto
5     .createHash('sha256')
6     .update(secret + patientId)
7     .digest('hex')
8 }
```

Listing 5.1 – Geração da chave de criptografia determinística

Assim, cifrar os dados médicos antes de enviá-los ao IPFS garante que as informações sensíveis permaneçam protegidas mesmo quando armazenadas na rede distribuída, atendendo aos requisitos de privacidade exigidos no contexto de dados de saúde.

Gerada a chave, o arquivo do prontuário médico é processado pela função `encryptContent` na linha 5 da Listagem 5.2. Ela recebe como parâmetros o *buffer* do arquivo original e a chave de criptografia previamente gerada, executando o processo de cifragem dos dados. O algoritmo de criptografia simétrica empregado é o *AES-256-GCM* (*Advanced Encryption Standard*).

```
1 let fileBuffer = fs.readFileSync(filePath)
```

```
2 // Encrypt content if encryption is enabled and patientId is provided
3 if (encryptFile && patientId) {
4   const encryptionKey = generateEncryptionKey(patientId)
5   const encryptionResult = encryptContent(fileBuffer, encryptionKey)
6   fileBuffer = encryptionResult.encryptedData
7 }
```

Listing 5.2 – Chamada da cifragem do conteúdo

Por se tratar de criptografia simétrica, a mesma chave cifra e decifra o conteúdo. Para não precisar armazená-la nem transmiti-la, ela é recriada de forma determinística tanto no envio quanto na recuperação, em duas etapas. Primeiro, a função `generateEncryptionKey` gera uma *passphrase* a partir de um *hash* SHA-256 sobre o *secret* de ambiente combinado com o *patientId*. Depois, a função `deriveKey` aplica PBKDF2 com 100.000 iterações sobre essa *passphrase* e um *salt* aleatório, chegando à chave de 256 *bits* usada pelo AES-256-GCM.

Como o *salt* e o vetor de inicialização (IV) são aleatórios a cada operação, precisam estar disponíveis na descriptografia. Em vez de transmitir a chave, o sistema concatena esses parâmetros ao conteúdo cifrado, formando um único pacote, conforme a Listagem 5.3.

```
1 const encryptedPackage = Buffer.concat([
2     salt,             // 16 bytes
3     iv,               // 16 bytes
4     authTag,          // 16 bytes
5     encryptedContent, // variable size
6 ])
```

Listing 5.3 – Montagem do pacote criptografado

Na recuperação, a função `decryptContent` extrai os 48 *bytes* iniciais para recuperar o *salt*, o IV e a *authentication tag*, rederiva a chave a partir do *patientId* e decifra o conteúdo, validando sua integridade pelo modo GCM.

Por fim, o arquivo criptografado é inserido na rede IPFS pela interface `fsUnix` (linha 1 da Listagem 5.4), utilizando o método `addBytes` (linha 2) para adicionar o *buffer* cifrado. Esse método gera um CID (*Content Identifier*) único, que funciona como endereço descentralizado para a recuperação posterior do registro médico armazenado na rede.

```
1     const { fsUnix } = await getHelia();
2     const cid = await fsUnix.addBytes(fileBuffer);
```

Listing 5.4 – Inserção do conteúdo cifrado na rede IPFS via Helia

5.2.4 Scripts de Migração

Os *scripts* de migração são os que fazem *deployment* dos contratos para a *blockchain* e garantem que as configurações de referências entre contratos estejam corretas.

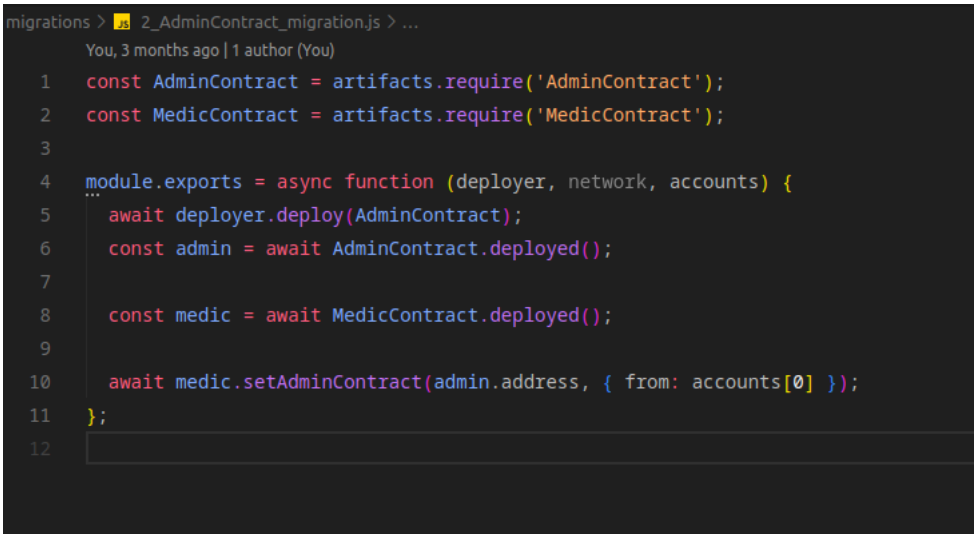


Figura 5.12 – Script de migração administrativo. Fonte: Elaboração Própria

A Figura 5.12 apresenta o script responsável pela migração e configuração inicial dos contratos inteligentes no ambiente Truffle. Nele, são carregadas as abstrações dos contratos, realizado o *deploy* do contrato administrativo e obtidas as instâncias implantadas. Por fim, o *script* associa o contrato médico ao contrato administrativo por meio do método `setAdminContract`, utilizando a conta padrão (accounts[0]) como origem da transação, garantindo a definição da autoridade administrativa no sistema.

5.2.5 *Endpoints* Gerais

Os *endpoints* estão na pasta `routes/`, organizados nos arquivos `adminRoutes.js`, `doctorRoutes.js`, `patientRoutes.js` e `fileRoutes.js`, facilitando a manutenção no *backend*. As Tabelas 5.1, 5.2, 5.3 e 5.4 apresentam os principais *endpoints* de cada categoria, descrevendo o método HTTP, a rota e os dados esperados em cada requisição.

A Tabela 5.1 reúne as operações administrativas, como o registro de médicos e pacientes; a Tabela 5.2, as funcionalidades disponíveis aos profissionais de saúde; a Tabela 5.3, as operações realizadas pelos pacientes sobre seus próprios registros; e a Tabela 5.4, o gerenciamento de arquivos via IPFS.

Tabela 5.1 – Endpoints - Gerenciamento Administrativo

Categoria	**Descrição**
Gerenciamento Administrativo	**POST** `/api/admin/doctors`: Registra um novo médico no sistema. Recebe *doctorId*, *name*, *specialization* e *licenseNumber* para autorização no *smart contract*.
	POST `/api/admin/patients`: Registra um novo paciente no sistema. Recebe *patientId*, *name*, *dateOfBirth*, *phoneNumber* e *emergencyContact*.

Tabela 5.2 – Endpoints - Funcionalidades dos Médicos

Categoria	**Descrição**
Funcionalidades dos Médicos	**GET** `/api/doctor/patients`: Retorna a lista de todos os pacientes ativos disponíveis para consulta médica.
	POST `/api/doctor/records`: Adiciona um novo registro médico para um paciente. Inclui *upload* de arquivo médico via *multipart/form-data* e armazenamento no IPFS.
	GET `/api/doctor/patients/{patientId}/exists`: Verifica se um paciente existe e está ativo no sistema antes de adicionar registros médicos.

Tabela 5.3 – Endpoints - Funcionalidades dos Pacientes

Categoria	**Descrição**
Funcionalidades dos Pacientes	**GET** `/api/patient/{patientId}`: Retorna informações detalhadas de um paciente específico, incluindo dados pessoais e status de ativação.
	GET `/api/patient/{patientId}/medical-records`: Retorna todos os registros médicos criados por médicos para o paciente, com opção de filtrar apenas registros ativos.
	GET `/api/patient/{patientId}/records`: Retorna registros médicos *auto-enviados* pelo próprio paciente através do *PatientContract*.
	GET `/api/patient/{patientId}/profile`: Retorna informações do perfil do paciente, incluindo dados pessoais e status de completude do perfil.
	POST `/api/patient/upload-record`: Permite que o paciente faça *upload* de seus próprios registros médicos, recebendo CID, *fileName*, *recordType* e *description*.

Tabela 5.4 – Endpoints - Gerenciamento de Arquivos

Categoria	**Descrição**
Gerenciamento de Arquivos	**GET** `/api/files/download/{cid}`: Realiza o *download* de arquivos médicos armazenados no IPFS através do *Content ID* (CID). Suporta arquivos criptografados com *query parameters* para *patientId* e *encrypted*.

6. AVALIAÇÃO

Este capítulo tem como objetivo descrever detalhadamente as etapas e os resultados obtidos por meio do desenvolvimento e implementação do *BlockHealth*. São descritos os procedimentos de validação, os testes realizados e a análise dos resultados experimentais que comprovam o funcionamento correto do sistema proposto.

6.1 Ambiente de Testes

Todos os testes foram executados em um computador utilizando os seguintes softwares e ferramentas:

- **Ganache**: *Blockchain* Ethereum local na porta 7545 com *network_id* 5777
- **Truffle Framework**: Versão 5.11.5 para compilação e execução de testes
- **Mocha e Chai**: Frameworks de teste para asserções e organização
- **Node.js**: Versão 14.x para execução da API REST
- **Helia/IPFS**: Implementação JavaScript do protocolo IPFS

A escolha de um ambiente local controlado permitiu a execução rápida sem custos de transação em redes públicas, e o isolamento total para validação de funcionalidades críticas sem interferências externas.

6.2 Testes Automatizados dos Contratos Inteligentes

A validação da *BlockHealth* foi conduzida por meio de testes automatizados organizados no diretório `test/`, refletindo a arquitetura dos contratos inteligentes.

Os testes são divididos por três arquivos principais. O `AdminContract.test.js` é responsável por validar operações administrativas, o `MedicContract.test.js` é focado em funcionalidades médicas e o `PatientContract.test.js`, dedicado a operações do paciente.

Os testes utilizaram o framework Truffle integrado com *Mocha* e *Chai*, foram executados sobre a *blockchain* Ethereum local Ganache. Cada teste foi inicializado através do hook `beforeEach()` o que garante o isolamento entre casos de teste e assegura que falhas de teste não afetem os demais.

A metodologia contempla testes positivos, que validam operações legítimas, e testes negativos, que verificam se o sistema rejeita adequadamente operações inválidas ou não autorizadas. Após cada operação, são realizadas asserções para verificar tanto o estado resultante na *blockchain* quanto os eventos emitidos pelos contratos.

6.2.1 Metodologia e Categorização dos Casos de Teste

A cobertura total compreende 53 casos de teste distribuídos entre os três contratos inteligentes, abrangendo todas as funcionalidades críticas e os principais cenários de falha do sistema *BlockHealth*.

Os casos de teste foram organizados em categorias funcionais por meio da função `describe()` do *Mocha*, permitindo agrupamento lógico, legibilidade e execução seletiva. A Tabela 6.1 apresenta a distribuição dos casos de teste por contrato e categoria.

Tabela 6.1 – Distribuição dos casos de teste por contrato e categoria

Contrato	**Categoria**	**Casos de Teste**
AdminContract	Implantação	2
	Gerenciamento de Médicos	5
	Gerenciamento de Pacientes	6
	Casos Extremos	1
MedicContract	Implantação	2
	Integração com AdminContract	2
	Autorização de Médicos	3
	Registros Médicos	7
	Casos Extremos	2
	Relacionamento Médico-Paciente	3
PatientContract	Implantação	3
	Gerenciamento de Perfil	4
	Gerenciamento de Registros Próprios	7
	Registros Adicionados por Médicos	2
	Controle de Acesso	2
	Casos Extremos	2
Total		**53**

As categorias contemplam dois tipos de abordagem. Os **testes de funcionalidade positiva** validam operações executadas por usuários autorizados com entradas válidas, verificando o estado resultante na *blockchain* e os eventos emitidos. Os **testes de**

controle de acesso e **casos negativos** asseguram que o sistema rejeita adequadamente operações realizadas por usuários sem permissão ou com parâmetros inválidos, verificando a ocorrência do *revert* nas transações.

Os **testes de integração entre contratos** verificam a comunicação entre os diferentes contratos inteligentes do sistema, em especial a dependência do `AdminContract` e do `MedicContract` em relação ao `AdminContract` para validação de autorizações. Por fim, os **testes de casos extremos** (*edge cases*) avaliam o comportamento do sistema diante de entradas vazias, valores extremos ou sequências incomuns de operações.

A execução da suíte de testes completa resultou em 53 casos aprovados em 12 segundos, sem nenhuma falha, conforme mostrado na Figura 6.1.

Figura 6.1 – Resultado da execução dos testes automatizados. Fonte: Elaboração Própria

6.3 Armazenamento dos dados das transações

A forma como os dados referentes às operações da *BlockHealth* são persistidos na cadeia de blocos é apresentada a seguir. Considerando que não foi empregada uma *blockchain* de produção, serão descritos o formato dos registros *on-chain* e os metadados associados à transação.

6.3.1 Exemplo de Registro de Paciente

Para demonstrar a funcionalidade operacional da arquitetura proposta analisa-se uma transação de registro de paciente executada pelo administrador do sistema que atua como agente *master* com privilégios administrativos.

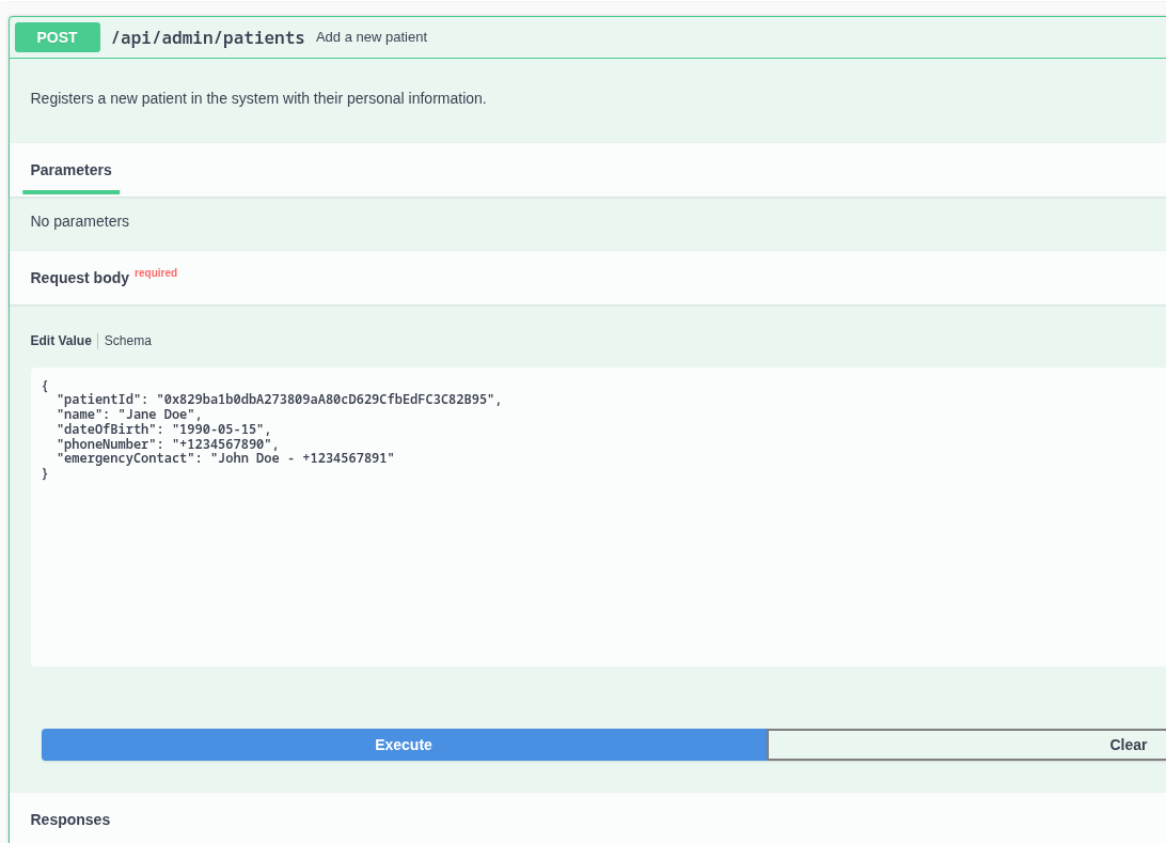


Figura 6.2 – Dados paciente. Fonte: Elaboração Própria

A Figura 6.2 apresenta os dados de entrada para o registro de um paciente no sistema. O campo `patientId` recebe o endereço Ethereum gerado na rede *blockchain*, que serve como identificador único e imutável do paciente. Os demais campos incluem informações pessoais como nome, data de nascimento, telefone e contato de emergência.

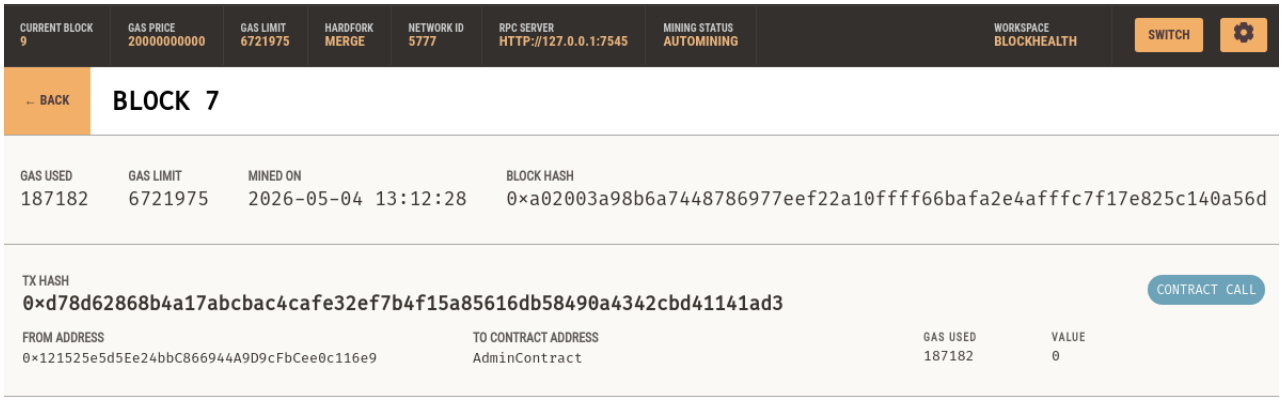


Figura 6.3 – Bloco contendo a transação de registro. Fonte: Elaboração Própria

Na Figura 6.3, observa-se que a operação resultou na chamada de contrato (*Contract Call*) registrada no bloco 7. A interface apresenta o *hash* do bloco e o *hash* da transação (*TX HASH*), que servem como identificadores únicos e imutáveis. O campo `TO CONTRACT ADDRESS` indica o endereço do contrato `AdminContract`, confirmando que a transação foi direcionada ao contrato responsável pelo gerenciamento de entidades do sistema.

O campo `From Address` indica o remetente da transação, neste caso o administrador do sistema, que possui privilégios para registrar pacientes. A transação foi assinada com a chave privada correspondente a este endereço.

6.3.2 Eventos Emitidos pela Transação

Figura 6.4 – Evento PatientRegistered emitido pelo contrato. Fonte: Elaboração Própria

A Figura 6.4 apresenta o evento emitido pelo contrato após a execução bem-sucedida da função de registro. Como visto anteriormente, os eventos em Solidity são mecanismos fundamentais para a notificação de aplicações externas, facilitando o monitoramento de mutações na *blockchain*.

O nome do evento foi `PatientRegistered` no `Event Name` definido no contrato para notificar aplicações externas.

O contrato de origem foi `AdminContract`, com a função de `registerPatient` e os dados do mesmo, confirmando que foi o responsável pela emissão.

6.4 Estrutura de Blocos na Blockchain

A Figura 6.5 apresenta uma sequência de blocos consecutivos minerados na rede Ganache durante a execução dos testes do sistema *BlockHealth*.

Figura 6.5 – Sequência de blocos na blockchain Ganache. Fonte: Elaboração Própria

Na Figura 6.5, observa-se a sequência de blocos. Esta visualização mostra a continuidade temporal da cadeia de blocos; cada novo bloco é adicionado sequencialmente, formando um registro cronológico imutável de todas as operações realizadas no sistema.

6.4.1 Propriedades Validadas Através da Análise de Blocos

A análise da estrutura de blocos e transações demonstra empiricamente as propriedades que justificam o uso da *blockchain* para a gestão de dados médicos.

Uma vez incluída no bloco 7, a transação de registro de paciente torna-se permanentemente imutável e inalterável, conforme apresenta na Figura 6.5. Qualquer alteração provocaria:

1. Alteração o *TX Hash* da transação original
2. Modificaria o *Block Hash* do bloco 7
3. Quebraria a cadeia de referências criptográficas com blocos subsequentes (8, 9, etc.)
4. Seria imediatamente detectável por todos os nós da rede através da verificação de *hashes* na rede

A imutabilidade garante que os registros médicos permaneçam íntegros, impedindo adulterações não autorizadas. Por meio dos identificadores criptográficos, como o endereço *hash* da transação, é possível rastrear completamente os aspectos da operação. A rastreabilidade se dá pela identificação do endereço da entidade que executou o registro, no caso, o administrador, além do evento propagado na função e do bloco imutável na cadeia de blocos.

6.5 Interação com a rede IPFS

Conforme apresentado na Seção 2.6, o IPFS foi adotado para o armazenamento distribuído e a recuperação de registros médicos. Essa arquitetura permite que os dados médicos sejam armazenados de forma redundante em múltiplos nós da rede, para eliminar pontos únicos de falha e garantir a resiliência. A seguir, descreve-se como a API desenvolvida interage com essa rede na inserção de prontuários médicos.

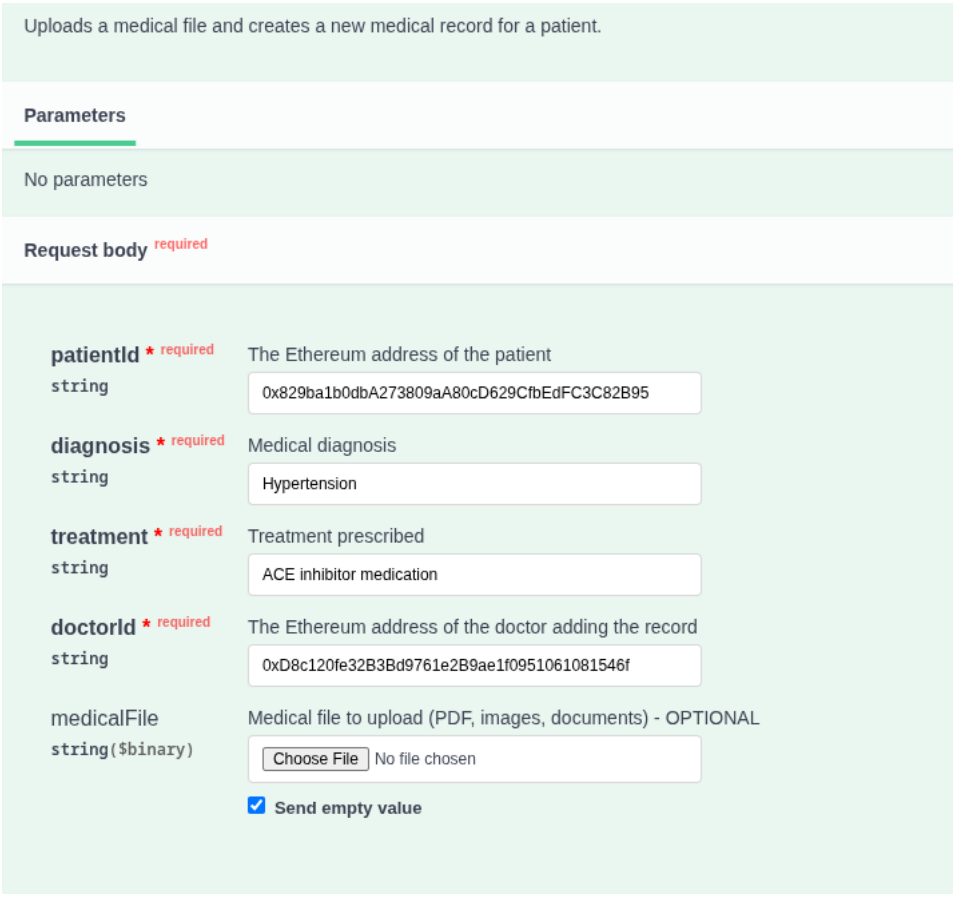


Figura 6.6 – Inserção de um prontuário médico, API. Fonte: Elaboração Própria

A Figura 6.6 apresenta a estrutura dos campos de requisição da API desenvolvida para a inserção de prontuários médicos. O processo operacional consiste em:

1. Recepção do arquivo no campo `medicalFile`, através da interface da API.
2. Aplicação prévia do algoritmo de criptografia de conteúdo para proteção dos dados (*content-encryption*).
3. Distribuição do arquivo criptografado nos nós da rede IPFS.
4. Retorno do *hash* identificador único do arquivo (*CID*) para futuras operações.

A descriptografia envolve um algoritmo que utiliza a chave de criptografia para a derivação da chave à criação do decifrador. A Figura 6.7 abaixo mostra o *endpoint* que realiza o processo de descriptografia, onde é passado como parâmetros o CID, patientId e um booleano.

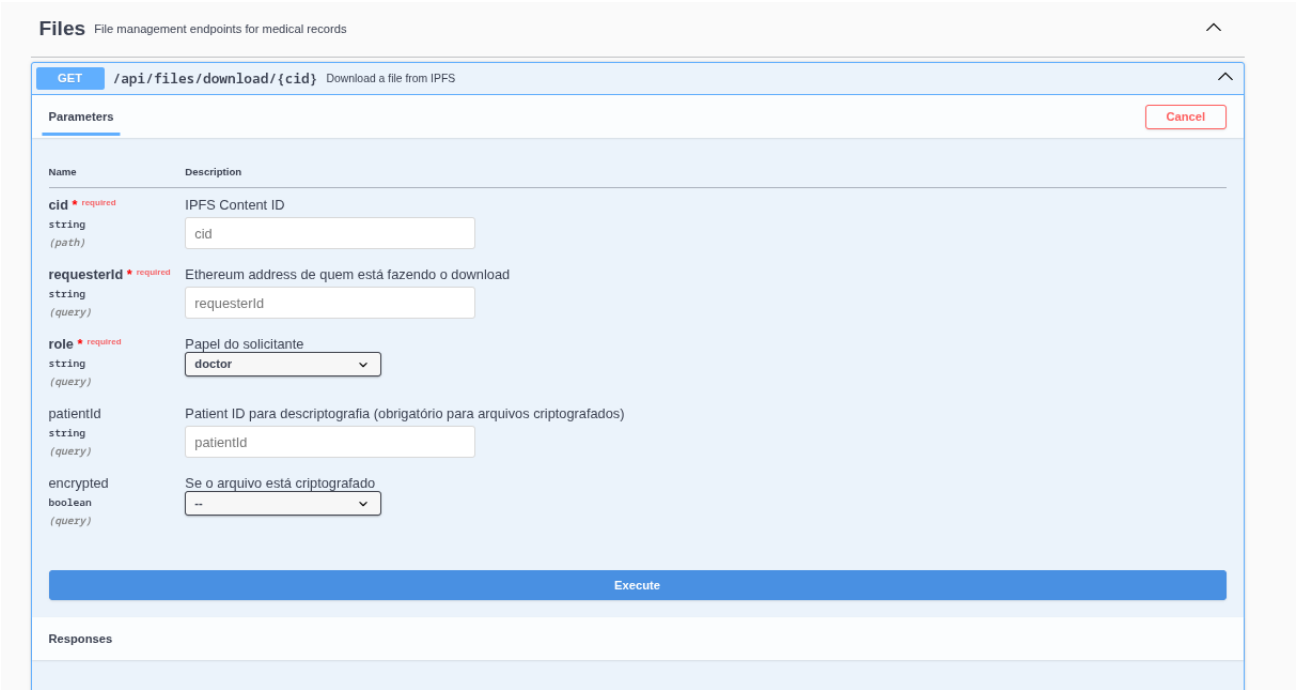


Figura 6.7 – Chamada de *endpoint* de descriptografia. Fonte: Elaboração Própria

A Figura 6.8, mostra o CID do registro médico inserido na rede IPFS criptografado.

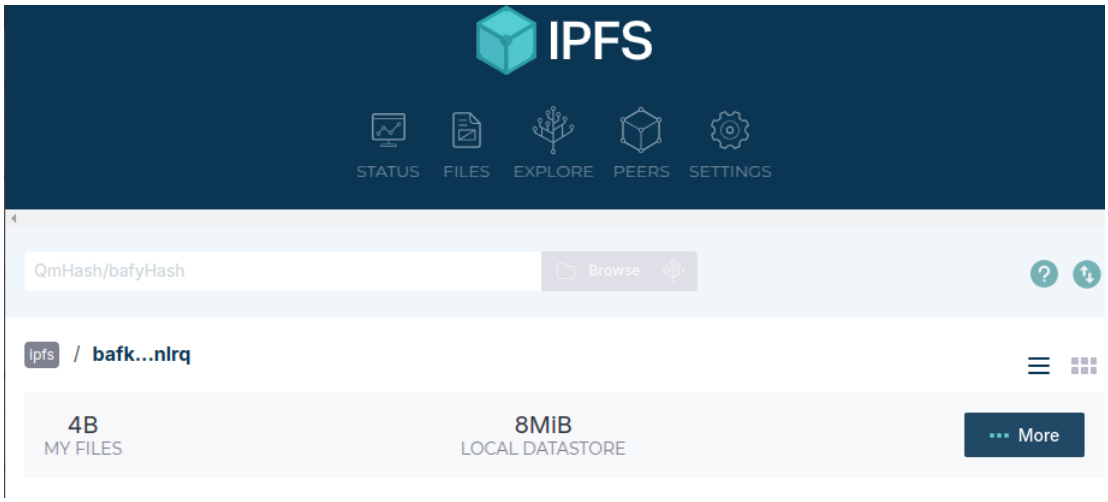


Figura 6.8 – Rede IPFS. Fonte: Elaboração Própria

6.6 Análise de Custo Computacional dos Contratos

Na arquitetura Ethereum, cada operação executada em um contrato inteligente consome uma quantidade específica de *gas*, como apresentado no Capítulo 3. Este mecanismo é fundamental para prevenir execuções infinitas e alocar recursos computacionais de forma eficiente na rede.

Analisou-se o custo computacional de uma das operações de maior carga de dados do sistema: a inserção de um prontuário médico, que envolve a validação de permissões, o processamento de dados e a interação com a rede IPFS para armazenamento do arquivo.

A Figura 6.9 apresenta os detalhes desta execução. A função executada é o `addMedicalRecordByAdmin` do contrato `MedicContract`, responsável por registrar um novo prontuário médico para um paciente específico.

Figura 6.9 – Detalhes de execução da função `addMedicalRecordByAdmin`. Fonte: Elaboração Própria

Como é possível observar na análise da transação, a execução desta função demanda um total de 310.793 unidades de *gas*. Este valor é influenciado por diversos fatores, incluindo a complexidade da função, o número de operações de leitura e escrita na *blockchain*, e a quantidade de dados processados. O contrato não apenas valida os endereços

do médico e paciente mas também registra variáveis tipo *string* como o CID do arquivo IPFS, nome do arquivo e o tratamento prescrito.

Ao comparar com o registro de paciente, que demandou 187.182 unidades de *gas*, constata-se que a inserção de um prontuário médico é significativamente mais custosa em termos computacionais, o que se deve à maior complexidade lógica e à maior quantidade de dados envolvidos na operação.

A Tabela 6.2 apresenta a comparação de custos entre as duas operações, obtida multiplicando-se o *gas* consumido pelo *gas price* de aproximadamente 1,4 *gwei* registrado no ambiente local Ganache, conforme observado na Figura 6.9.

Tabela 6.2 – Comparação de custos de transação

Função	**Gas Utilizado**	**Custo (ETH)**	**Custo (USD)**
Registro de Paciente	187.182	0,00026	$0,62
Inserção de Prontuário	310.793	0,00043	$1,02

Nota: valores calculados com o *gas price* de aproximadamente 1,4 *gwei* (1.399.000.623 *wei*) registrado no ambiente local Ganache (Figura 6.9) e cotação de 1 ETH = US$ 2.357,00, referente a maio de 2026. Os valores em dólar são ilustrativos, uma vez que o ambiente Ganache não opera com ETH de valor econômico real.

Ressalta-se que o *gas price* do Ganache é definido de forma arbitrária pela ferramenta e não corresponde ao de uma rede real. Na *mainnet* Ethereum, esse valor oscila conforme o congestionamento da rede em julho de 2026, a média era em torno de 0,3 *gwei*, chegando a um valor de dois dígitos nos momentos de maior demanda (Etherscan, 2026). Em redes de segunda camada (*Layer 2*), como a Polygon, os valores são ainda menores. Ou seja, mesmo que os custos fiquem altos em momentos de pico na *mainnet*, a arquitetura proposta pode ser otimizada com o uso de redes alternativas, sem abrir mão da segurança e da rastreabilidade.

6.6.1 Considerações Finais

A análise detalhada da estrutura de blocos na *blockchain* Ganache, comprova experimentalmente que todas as operações do sistema *BlockHealth* são registradas permanentemente em blocos imutáveis e criptograficamente seguros.

Por outro viés, considerando uma *blockchain* de produção como a Ethereum, seria mais custoso, pois, como visto, as transações dependem de *gas* e estão sujeitas a volatilidade do mercado e ao congestionamento da rede. Isso indica que seria necessária uma avaliação melhor para uma aplicação de larga escala.

6.7 Pontos de Melhoria

Um fator que pode ser melhorado consiste na implementação de uma arquitetura organizacional mais robusta. Atualmente, o sistema opera com três níveis básicos de permissão de usuários; porém, a incorporação de um novo componente para as organizações médicas `Organization` representaria um avanço na gestão hierárquica de instituições de saúde, tornando, assim, o sistema mais flexível.

A implementação do componente `Organization` contemplaria a criação de entidades organizacionais que agregariam diferentes tipos de instituições médicas, hospitais, unidades básicas de saúde, clínicas especializadas, entre outros. Cada organização teria um administrador responsável pela gestão dos usuários e recursos específicos de sua instituição.

Outro fator de melhoria seria na implementação do banco de dados NoSQL como MongoDB e Redis, a fim de armazenar dados de usuários para o controle do acesso de fluxo de permissão. Esta abordagem aprimoraria o sistema de controle de acesso e reduziria a latência das operações. Uma estratégia semelhante foi adotada por (AGOSTINHO et al., 2019), que emprega o MongoDB para o armazenamento otimizado de chaves públicas.

7. CONCLUSÃO

A tecnologia *blockchain*, juntamente com os *smart contracts*, emerge como uma solução para o problema de segurança, privacidade e interoperabilidade que a gestão de dados médicos possui. Diante desses desafios, este trabalho propôs o desenvolvimento da aplicação *BlockHealth*, uma solução descentralizada baseada em *smart contracts* para o compartilhamento seguro de dados de saúde, projetada para automatizar processos burocráticos, com ênfase na integridade dos dados e para proporcionar controle granular de acesso às informações médicas.

A implementação da *BlockHealth* seguiu uma abordagem que integra a *blockchain* da rede Ethereum com o protocolo *InterPlanetary File System* (IPFS) para o armazenamento distribuído de arquivos médicos. Os contratos inteligentes foram desenvolvidos na linguagem Solidity que lida com a lógica de negócio para diferentes tipos de usuários, implementando mecanismos de controle de acesso específicos. A arquitetura modular do sistema, com separação clara entre camadas de apresentação, lógica de negócio e persistência, facilitou o desenvolvimento e garantiu a escalabilidade da solução.

Os testes realizados em ambiente local utilizando o Ganache conseguiram atender aos requisitos de segurança, transparência e controle de acesso estabelecidos. A integração com a IPFS, com a combinação do *content-encryption*, provou ser eficaz para o armazenamento de arquivos médicos de forma descentralizada, enquanto os *smart contracts* garantiam a imutabilidade dos registros e transações.

Dessa forma, este trabalho contribui para o avanço da aplicação de tecnologias *blockchain* na área da saúde, demonstrando que há uma viabilidade técnica de soluções descentralizadas para gestão de dados médicos. A *BlockHealth* representa um passo importante no desenvolvimento de sistemas mais seguros e transparentes, estabelecendo uma base sólida para futuras pesquisas na tecnologia e novos desenvolvimentos nesta área.

7.1 Trabalhos Futuros

Como trabalhos futuros, uma direção importante é a implementação em *testnets* públicas para validar a escalabilidade e a performance em cenários reais. Também se identificam oportunidades de aprimoramento do *BlockHealth* e de desenvolvimento de novas aplicações utilizando essas tecnologias. Adicionalmente, a expansão de funcionalidades para a inclusão de recursos como conformidade automatizada com regulamentações de proteção de dados e integração com sistemas hospitalares existentes representa um caminho promissor para futuras pesquisas.

REFERÊNCIAS BIBLIOGRÁFICAS

AGOSTINHO, B. et al. Unificação de dados de saúde através do uso de blockchain e smart contracts. In: *Anais da XV Escola Regional de Banco de Dados*. Porto Alegre, RS, Brasil: SBC, 2019. p. 31–40. ISSN 2595-413X. Disponível em: <https://sol.sbc.org.br/index.php/erbd/article/view/8476>.

AHAMED, N. N. A build and deploy ethereum smart contract for food supply chain management in truffle - ganache framework. In: . [S.l.: s.n.], 2023.

CHITTODA, J. *Mastering Blockchain Programming with Solidity: Write production-ready smart contracts for Ethereum blockchain with Solidity*. Birmingham: Packt Publishing, 2019. ISBN 9781839218262.

CHONDROGIANNIS, E. et al. Using blockchain and semantic web technologies for the implementation of smart contracts between individuals and health insurance organizations. *Blockchain: Research and Applications*, v. 3, n. 2, p. 100049, 2022. ISSN 2096-7209. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2096720921000440>.

Ethereum Foundation. *The Merge*. 2022. <https://ethereum.org/roadmap/merge/>. Acesso em: 9 jul. 2026.

Etherscan. *Ethereum Gas Tracker*. 2026. <https://etherscan.io/gastracker>. Acesso em: 7 jul. 2026.

HALEEM, A. et al. Blockchain technology applications in healthcare: An overview. *International Journal of Intelligent Networks*, v. 2, p. 130–139, 2021. ISSN 2666-6030. Disponível em: <https://www.sciencedirect.com/science/article/pii/S266660302100021X>.

IPFS / HELIA CONTRIBUTORS. *Helia — A lean, modular and modern implementation of IPFS in JavaScript*. 2025. <https://ipfs.github.io/helia/>. Acesso inicial em jun. 2026.

IPFS Docs. *IPFS Docs*. n.d. Acesso em: 10 de agosto de 2025. Disponível em: <https://docs.ipfs.tech>.

IPFS Docs. *Privacy and encryption*. n.d. Acesso em: 10 de agosto de 2025. Disponível em: <https://docs.ipfs.tech/concepts/privacy-and-encryption/#encryption>.

JAIMAN, V.; UROVI, V. A consent model for blockchain-based health data sharing platforms. *IEEE Access*, v. 8, p. 143734–143745, 2020.

KHALID, M. I. et al. A comprehensive survey on blockchain-based decentralized storage networks. *IEEE Access*, v. 11, p. 10995–11015, 2023.

KHATOON, A. A blockchain-based smart contract system for healthcare management. *Electronics*, v. 9, n. 1, 2020. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/9/1/94>.

MOOSAVI, N. et al. Blockchain technology, structure, and applications: A survey. *Procedia Computer Science*, v. 237, p. 645–658, 2024. ISSN 1877-0509. International Conference on Industry Sciences and Computer Science Innovation. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050924011669>.

MUKHOPADHYAY, M. *Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity*. 1. ed. Birmingham: Packt Publishing Ltd, 2018. ISBN 978-1-78847-304-0.

NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. <https://bitcoin.org/bitcoin.pdf>. Acesso em: 8 jul. 2026.

ROUHANI, S.; DETERS, R. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access*, v. 7, p. 50759–50779, 01 2019.

SAMUEL, O. W. et al. Activity recognition based on pattern recognition of myoelectric signals for rehabilitation. In: KHAN, S. U.; ZOMAYA, A. Y.; ABBAS, A. (Ed.). *Handbook of Large-Scale Distributed Computing in Smart Healthcare*. Cham: Springer, 2017, (Scalable Computing and Communications). p. 427–442.

SEGAL, G. et al. A blockchain-based computerized network infrastructure for the transparent, immutable calculation and dissemination of quantitative, measurable parameters of academic and medical research publications. *DIGITAL HEALTH*, v. 9, p. 20552076231194851, 2023. Disponível em: <https://doi.org/10.1177/20552076231194851>.

SHOJAEI, P.; VLAHU-GJORGIEVSKA, E.; CHOW, Y.-W. Security and privacy of technologies in health information systems: A systematic literature review. *Computers*, v. 13, n. 2, p. 41, 2024.

TANENBAUM, A. S.; STEEN, M. van. *Distributed Systems: Principles and Paradigms*. 3. ed. [S.l.]: Maarten van Steen, 2016.

WANG, Z. et al. Sharing service in healthcare systems: A recent survey. *Omega*, v. 129, p. 103158, 2024. ISSN 0305-0483. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305048324001233>.

YAMAGATA, D. *Understand EVM Opcodes, Write Better Smart Contracts*. 2022. Medium. Acesso em: 05 dez. 2025. Disponível em: <https://medium.com/@danielyamagata/understand-evm-opcodes-write-better-smart-contracts-e64f017b619>.

ZAGHLOUL, E.; LI, T.; REN, J. Security and privacy of electronic health records: Decentralized and hierarchical data sharing using smart contracts. In: *2019 International Conference on Computing, Networking and Communications (ICNC)*. [S.l.: s.n.], 2019. p. 375–379.