INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE SANTA CATARINA

CAMPUS SÃO JOSÉ

GUILHERME DA SILVA DE MEDEIROS

DESENVOLVIMENTO DE UMA API RESTFUL E INTEGRAÇÃO COM BLOCKCHAIN ETHEREUM PARA APLICAÇÃO DE SMARTGRID

SÃO JOSÉ

2025

Guilherme da Silva de Medeiros

Desenvolvimento de uma API RESTful e integração com blockchain Ethereum para aplicação de smartgrid

Monografia apresentada ao Curso de Engenharia de Telecomunicações do Instituto Federal de Santa Catarina, para a obtenção do título de bacharel em Engenharia de Telecomunicações.

Área de concentração: Telecomunicações

Orientador: Prof. Diego da Silva de Medeiros, Dr.

São José

2025

Guilherme da Silva de Medeiros

Desenvolvimento de uma API RESTful e integração com blockchain Ethereum para aplicação de smartgrid

Monografia apresentada ao Curso de Engenharia de Telecomunicações do Instituto Federal de Santa Catarina, para a obtenção do título de bacharel em Engenharia de Telecomunicações.

São José, 28 de Agosto de 2025.

Documento assinado digitalmente
DIEGO DA SILVA DE MEDEIROS
Data: 18/08/2025 14:03:04-0300
CPF: ***.242.729-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Diego da Silva de Medeiros, Dr.

Instituto Federal de Santa Catarina

Documento assinado digitalmente
EVANASKA MARIA BARBOSA NOGUEIRA
Data: 18/08/2025 14:30:49-0300
CPF: ***.376.004-**
Verifique as assinaturas em https://v.ufsc.br

Professora Evanaska Maria Barbosa Nogueira, Dra.

Instituto Federal de Santa Catarina

Documento assinado digitalmente
RAMON MAYOR MARTINS
Data: 18/08/2025 14:11:56-0300
CPF: ***.866.756-**
Verifique as assinaturas em https://v.ufsc.br

Professor Ramon Mayor Martins, Dr.

Instituto Federal de Santa Catarina

Dedico este trabalho à minha família, que nunca desistiu de me apoiar.

AGRADECIMENTOS

Agradeço a minha família pela paciência. Agradeço aos meus colegas pelas risadas. Agradeço a mim mesmo por nunca desistir.

"All that is gold does not glitter
Not all those who wander are lost
The old that is strong does not wither
Deep roots are not reached by the frost
From the ashes a fire shall be woken
A light from the shadows shall spring
Renewed shall be blade that was broken
The crownless again shall be king"
(J. R. R. Tolkien, 1954)

RESUMO

Este trabalho propõe uma solução tecnológica para facilitar transações de energia elétrica no mercado livre brasileiro por meio da integração entre redes inteligentes (smart grids) e tecnologia blockchain. ntando com um levantamento bibliográfico e análise do setor energético nacional, abordando o crescimento da demanda elétrica e a expansão do mercado livre de energia. Foi explorado o conceito de smart grids como ecossistemas tecnológicos capazes de suportar a geração, distribuição, monitoramento e comercialização de energia com maior eficiência e segurança. Na sequência, investigou-se o papel da blockchain como ferramenta capaz de garantir integridade, rastreabilidade e imutabilidade nas transações energéticas peer-to-peer. Como aplicação prática, foi desenvolvida uma API RESTful conectada a uma instância Ethereum local, executada em contêiner Docker, capaz de criar contas, consultar saldos e realizar transações simuladas de energia. A arquitetura empregada foi modular e em microsserviços, utilizando autenticação criptográfica e validação por consenso. Os resultados demonstraram ganhos em auditabilidade, eliminação de intermediários, segurança dos dados e flexibilidade para expansão futura da solução. Apesar de desafios como latência e custos de armazenamento, estratégias foram adotadas para otimizar desempenho. Como proposta de continuidade, recomenda-se o desenvolvimento de contratos inteligentes específicos e interfaces web para facilitar o uso por usuários comuns. A conclusão aponta que a integração entre smart grids e blockchain representa um avanço significativo rumo à modernização e descentralização do setor energético brasileiro.

Palavras-chave: smart grid, blockchain, energia elétrica, mercado livre, API, transações seguras.

ABSTRACT

This study proposes a technological solution to enable secure energy transactions in Brazil's free energy market through the integration of smart grids and blockchain technology. The methodology included literature review and analysis of the national energy sector, emphasizing rising electricity demand and the growing relevance of decentralized energy trading. Smart grids were examined as interconnected technologies that enhance the generation, distribution, monitoring, and commercialization of energy. Blockchain was investigated as a mechanism to ensure transaction integrity, traceability, and immutability in peer-to-peer energy exchanges. A RESTful API was developed and integrated into a local Ethereum instance running in a Docker container, allowing account creation, balance queries, and simulated energy transactions. The architecture is based on a microservices model, using asymmetric cryptography and consensus validation. Results highlight improvements in auditability, data security, and elimination of centralized intermediaries. Despite challenges such as latency and storage costs, mitigation strategies were implemented to optimize performance. For future enhancements, the study recommends developing specialized smart contracts and a user-friendly web application interface. The conclusion reinforces that integrating smart grids and blockchain marks a significant step toward modernizing and decentralizing Brazil's energy sector.

Keywords: smart grid, blockchain, energy market, decentralized systems, API, secure transactions.

LISTA DE ILUSTRAÇÕES

Figura 1 – Aumento do consumo de energia elétrica 13
Figura 2 – Diagrama da CCEE sobre o mercado livre de energia 15
Figura 3 – Diagrama da CCEE sobre o aumento de uso de mercado livre de energia no Brasil 16
Figura 4 – Diagrama detalhando o funcionamento de uma *blockchain* 23
Figura 5 – Exemplo de transação utilizando *blockchain* 24
Figura 6 – Exemplo de arquitetura para ampla aplicação de *blockchain* 27
Figura 7 – Métodos de uma API Restful 36
Figura 8 – Sistemas que compõem a arquitetura utilizada 43
Figura 9 – Arquitetura geral implementada 45
Figura 10 – Criação de conta 46
Figura 11 – Transação 46

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVOS 12
1.1.1 Objetivo geral 12
1.1.2 Objetivos específicos 12

2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 SOBRE O MERCADO DE ENERGIA ELÉTRICA NO BRASIL 13
2.2 *SMART GRID* 14
2.2.1 Características de *Smart Grids* 15
2.2.2 Segurança cibernética em *Smart Grids* 19
2.3 BLOCK CHAIN 21
2.3.1 Aplicações de *Blockchains* 24
2.3.1.1 Blockchains em Sistemas de Transporte 24
2.3.1.2 Blockchain na Internet das Coisas 26
2.4 *BLOCKCHAINS* EM *SMART GRIDS* 28
2.4.1 Casos de uso de *blockchains* em *smart grids* 28
2.4.1.1 Compra de energia peer-to-peer 28
2.4.1.2 Técnicas de Segurança e Preservação de Privacidade em Smart Grids com Blockchain 30
2.4.1.3 Manutenção preditiva e segurança de equipamentos em smart grids . . . 33
2.4.2 Estudo e Comparação de Estratégias para Implementação de *Blockchain* em *Smart Grids* 34
2.4.2.1 Implementação com blockchain Ethereum Local usando Ganache 34
2.4.2.2 Desenvolvimento de blockchain Própria com Estrutura Modular 35
2.4.2.3 Uso de Plataformas blockchain Permissionadas 35
2.5 *APPLICATION PROGRAMMING INTERFACE* (API) 36
2.5.1 Modelos de Implementação de APIs 37
2.5.2 REST: Transferência Representacional de Estado 37
2.5.3 Benefícios das APIs REST 38
2.6 CONTÊINERES E DOCKER: CONCEITOS, FUNCIONAMENTO E APLICAÇÕES 39
2.6.1 Conceito de Contêineres 39
2.6.2 Docker: Plataforma de Contêineres 39

3 DESENVOLVIMENTO DA APLICAÇÃO 41

3.1 ESTUDO DE *SMART GRIDS* E MAPEAMENTO CONTEXTUAL DO SETOR ELÉTRICO NACIONAL . . . 41
3.2 DETECÇÃO DAS PRINCIPAIS DEMANDAS DE DESENVOLVIMENTO PARA A APLICAÇÃO DE ENERGIA INTELIGENTE . . . 42
3.2.1 Negociação Descentralizada de Energia com *Blockchain* . . . **42**
3.3 ARQUITETURA DA SOLUÇÃO IMPLEMENTADA . . . 43
3.4 APLICAÇÃO DETALHADA . . . 44
3.4.1 Visão Geral e arquitetura . . . **44**
3.4.2 Funcionalidades . . . **45**
3.4.3 *Endpoints* Detalhados da API *RESTful* . . . **46**
3.4.3.1 Contas . . . *47*
3.4.3.2 Transações . . . *48*

4 CONCLUSÕES . . . **49**
4.1 DESEMPENHO E ESCALABILIDADE . . . 49
4.2 VANTAGENS DA IMPLEMENTAÇÃO . . . 50
4.2.1 Transparência e Auditabilidade . . . **50**
4.2.2 Segurança e Integridade de Dados . . . **50**
4.2.3 Potencial para Mercados de Energia Descentralizados . . . **50**
4.3 ANÁLISE TÉCNICA DA IMPLEMENTAÇÃO . . . 51
4.3.1 Avaliação da Arquitetura . . . **51**
4.4 QUANTO A APLICABILIDADE DA IMPLEMENTAÇÃO COMO PROVA DE CONCEITO . . . 51
4.5 RECOMENDAÇÕES PARA EVOLUÇÃO FUTURA . . . 52
4.6 CONSIDERAÇÕES FINAIS . . . 52

Referências . . . **53**

APÊNDICE A – CÓDIGO DA FERRAMENTA DESENVOLVIDA: **55**

1 INTRODUÇÃO

Analisando um histórico recente de uso de energia elétrica (CCEE, 2024), é possível notar um alto crescimento de uso de energia por todo o país. De setores como a agricultura, indústria e tecnologia, principalmente grandes projetos de inovação tecnológica tendem com o passar dos anos. Esse crescimento é evidenciado pelos estudos que a Câmara de Comercialização de Energia Elétrica fez em 2024, mostrando que, por isso, a utilização do mercado livre de energia vem aumentando consideravelmente.

O mercado livre de energia (CCEE, 2025, 2024) é um modo de geração e transmissão de energia feito de maneira livre, de um produtor para um consumidor, por fora de órgãos públicos ou terceirizados de geração de energia elétrica. Ele possui regulamentação no país, que prevê como esse mercado deve funcionar, estipula regras e define maneiras e tecnologias que possibilitam essa troca *peer-to-peer* deste bem tão necessário para a sociedade.

As *smart grids* (FALCÃO, 2010) surgem como uma possível solução para vários dos problemas enfrentados quando a intenção é a implementação de um mercado livre de energia. Em tradução livre, a *Energia Inteligente*, não é uma tecnologia em si, mas sim uma porção de tecnologias que podem possibilitar este tipo de fornecimento, manutenção e mercado. Em *smart grids* (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011) se define formas de geração de energia elétrica, métodos de sensoriamento para a garantia da qualidade energética, formas de transmissão de dados para áreas chaves nas quais essas fazendas de energia seriam estabelecidas, maneiras de transmissão de energia da produção ao usuário consumidor e até formas práticas de se fazer as transações monetárias que categorizam a compra e venda de energia elétrica.

Quando se trata de compra e venda de energia, uma das demandas da aplicação de *smart grids*, uma possibilidade de criar transações íntegras e seguras, garantindo rastreabilidade e imutabilidade é a utilização de uma *blockchain* (SUNNY et al., 2022; ALLADI et al., 2019).

Blockchains foram definidas pela primeira vez por Satoshi Nakamoto (NAKAMOTO, s.d.) (pseudônimo) no seu manifesto em 2008 detalhando uma solução matemática para a troca de dinheiro utilizando uma tecnologia computacional baseada em blocos que se conectam e se analisam para garantir a integridade das transações que cada bloco representa. Definir e explicar uma *blockchain* é complicado e será feito no capítulo de Fundamentação Teórica, por aqui, basta saber que é um método amplamente testado (AHRAM et al., 2017) e com uma arquitetura complexa que garante a integridade dos dados, assim como a privacidade e segurança de quem realiza as transações se assim for

desejado.

Entretanto, desenvolvimento de aplicações utilizando *blockchain* pode ser complicado (ALLADI et al., 2019). Existem uma porção de *blockchains* no mercado, além de existir a possibilidade de um desenvolvedor implementar sua própria, caso seja de seu desejo. Formas de se comunicar com as *blockchains* são várias e neste trabalho serão detalhadas algumas arquiteturas que possibilitariam trocar informações com uma *blockchain* utilizando um serviço de Interface de Programação de Aplicações, uma API.

Como parte dos objetivos deste trabalho é a implementação de uma solução para compôr uma complexa arquitetura que possibilite transações de energia elétrica em mercado livre utilizando no contexto de *smart grids*, foi decidido implementar uma API capaz de se comunicar com uma *blockchain*, fazendo transações seguras, que são guardadas na *blockchain*.

Foi escolhida uma arquitetura de microsserviço, com a *blockchain* sendo executada em container *docker* (que será melhor explicado também na sessão de fundamentação teórica), com a API rodando em paralelo, fazendo requisições para a *blockchain*.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Desenvolver uma ferramenta capaz de realizar transações energéticas no contexto de *smart grids* (energia inteligente), afim de atender uma das várias demandas existentes no mercado de energia inteligente. Além disso, é objetivo deste trabalho fazer um estudo aprofundado sobre *smart grids*, suas aplicações e desafios de implementação.

1.1.2 Objetivos específicos

- Estudar a tecnologia de *smart grid*, suas aplicações e desafios de implementação;
- Verificar demandas necessárias para a ativação ou expansão desse tipo de mercado;
- Analisar, como forma de motivação deste trabalho, o mercado de energia no Brasil e a aplicação já existente dos temas centrais deste estudo.
- Explicar como as transações de energia via *blockchain* são necessárias para a aplicação de *smart grids*
- Criar uma arquitetura de *software* no qual seria possível desenvolver uma ferramenta onde as transações de energia sejam possíveis, seguras e íntegras.
- Desenvolver uma ferramenta capaz de realizar essas transações, apresentando uma solução para esta demanda.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 SOBRE O MERCADO DE ENERGIA ELÉTRICA NO BRASIL

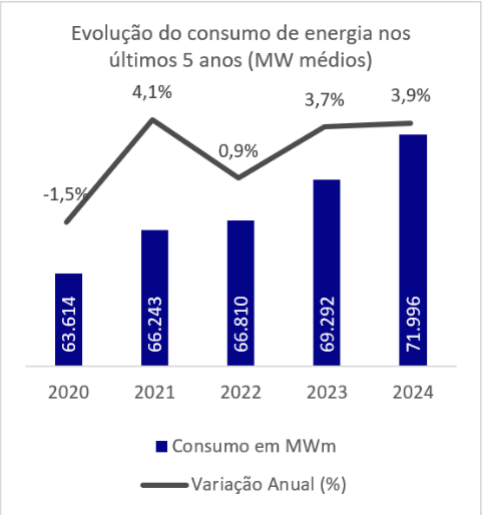
A energia elétrica representa um dos pilares fundamentais da sociedade humana, impulsionando simples dispositivos eletrônicos, complexos sistemas industriais, gigantescos servidores que contém a maioria da informação criada pela humanidade, serviços de importância crítica para o funcionamento da sociedade, entre outras aplicações.

A distribuição eficiente desta energia tem sido um constante desafio, exigindo continuamente inovações tecnológicas para atender à crescente demanda. Ao longo dos séculos XX e XXI, foi testemunhada uma notável evolução dos sistemas de distribuição de energia, partindo de simples redes centralizadas até as atuais infraestruturas distribuídas e interconectadas (FALCÃO, 2010).

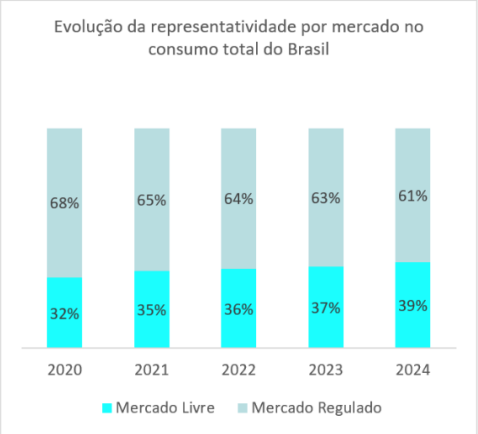
Os sistemas de energia estão em constante mudanças para a garantia desses serviços, desde 2020 (CCEE, 2024) o consumo de energia no Brasil está em aumento constante, em 2024 ultrapassou pela primeira vez a marca de 70 mil megawatts médios. Segundo a CCEE (Câmara de Comercialização de Energia Elétrica), o país consumiu 3,9 por cento a mais do que em 2023 tendo como o calor e o aumento da atividade industrial como maiores razões do aumento do consumo.

Figura 1 – Aumento do consumo de energia elétrica

(a) Aumento da energia elétrica no Brasil



(b) Aumento da energia fornecida pelo mercado livre



Fonte: CCEE2024

A Figura 1 evidencia o aumento de energia elétrica previamente mencionado, assim como uma outra informação: apesar do consumo regulado ainda representar a maior parte

da energia consumida pelo Brasil em 2024, a quantidade de energia consumida que vem do mercado livre vem aumentando com o passar dos anos.

O mercado livre de energia (CCEE, 2025) (também conhecido como ambiente de contratação livre) é a modalidade de negociação em que o consumidor é capaz de escolher o fornecedor de energia elétrica, assim como negociar preços e condições do serviço. Desde Janeiro de 2024 todos os consumidores de alta e média tensão podem optar pela migração para essa modalidade, no qual o consumidor paga uma fatura referente ao serviço de distribuição para a concessionária local (regulada) e outra com os valores de consumo para o gerador ou produtor independente de energia.

A Figura 2 ajuda a compreender melhor o sistema de ambiente de contratação livre de energia.

Ainda sobre o aumento de consumo de energia elétrica (CCEE, 2024), a CCEE acompanha o crescimento do mercado livre de energia no país considerando 15 setores da economia, com o saneamento básico, serviços e comércio mostrando o maior aumento em 2024, muito impulsionados pela migração de algumas empresas do setor para o ambiente de livre mercado, no caso do saneamento básico (aumento de 47,9 por cento do uso do mercado livre de energia, o maior entre os setores estudados) é resultado do marco legal do Saneamento Básico(CCEE, 2024).

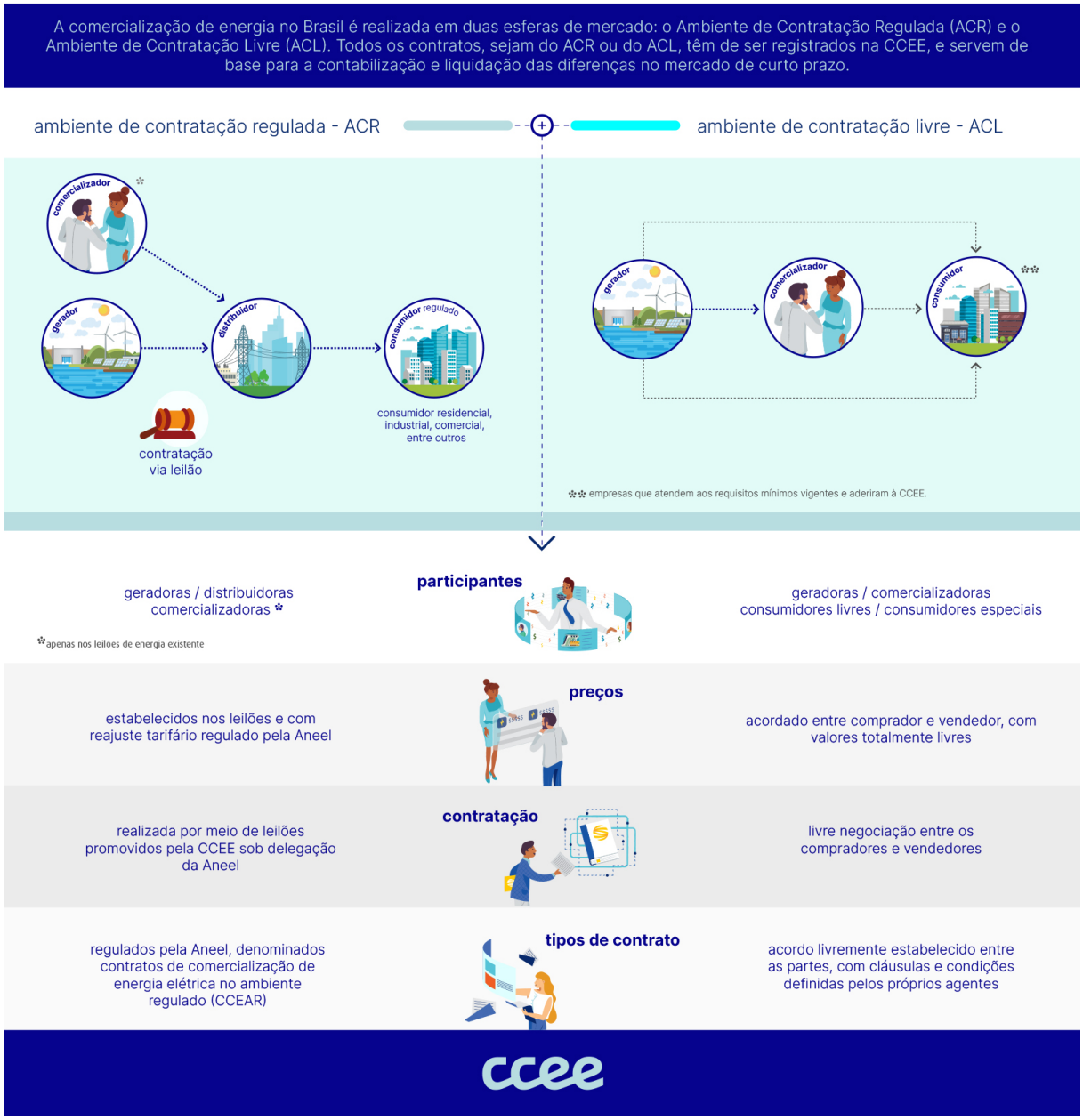
É importante salientar que segundo a CCEE a geração da energia usada no mercado livre de energia é em grande maioria oriunda de fontes renováveis como parques eólicos, fotoelétricos ou biomassa, por isso não é possível observar piora no quadro de energia renovável do Brasil, a figura 3 demonstra que não existe tendência de crescimento do uso de energia não renovável desde 2020, quando o mercado livre de energia começou a crescer.

2.2 *SMART GRID*

Os dados da CCEE são significativos para demonstrar que o aumento do consumo de energia elétrica, com o aumento do uso de sistemas de energia elétrica tornam-se necessárias mudanças que tragam modernizações das tecnologias de geração, transmissão, distribuição e uso final da energia elétrica, visando abordar questões diversas como mudanças climáticas (e os impactos negativos que advém de fenômenos extremos e altas temperaturas sobre a transmissão de energia), envelhecimento das instalações elétricas e a necessidade de proporcionar ao usuário final maior participação de decisões do sistema elétrico que fornece a energia que ele consome (FALCÃO, 2010).

Uma nova concepção da maneira como geramos energia elétrica atende pelo nome de *smart grid*, que deve ser entendido mais como conceito do que como tecnologia ou equipamento específico (FALCÃO, 2010). Entende-se como *smart grid* uma forma de

Figura 2 – Diagrama da CCEE sobre o mercado livre de energia



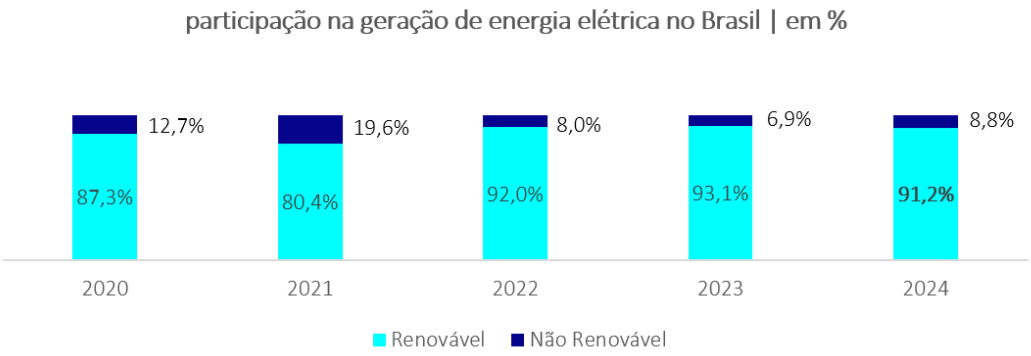
Fonte: CCEE2025

utilização de tecnologia de automação, computação e comunicação para melhor monitoramento e controle da rede elétrica em um mundo que consome cada vez mais energia. O objetivo do empenho dessas tecnologias é permitir a implantação de melhores estratégias de otimização da rede de forma eficiente e transparente.

2.2.1 Características de *Smart Grids*

Algumas característica geralmente atribuídas à *smart grids* são segundo Falcão (2010) e Paul (2014):

Figura 3 – Diagrama da CCEE sobre o aumento de uso de mercado livre de energia no Brasil



Fonte: CCEE2024

1. Autorreparação: capacidade de detectar, analisar e responder automaticamente a falhas na rede elétrica, promovendo sua recuperação de forma autônoma e minimizando interrupções no fornecimento de energia.

2. Empoderamento do consumidor: integração dos equipamentos e dos padrões de comportamento dos consumidores nos processos de planejamento e operação da rede, promovendo sua participação ativa no gerenciamento energético.

3. Tolerância a ataques externos: habilidade de mitigar e resistir a ameaças físicas e cibernéticas, garantindo a segurança e a confiabilidade da infraestrutura elétrica.

4. Qualidade da energia: fornecimento de energia elétrica com níveis adequados de qualidade, compatíveis com as exigências das sociedades modernas.

5. Integração de diversas fontes e demandas: capacidade de incorporar, de forma ágil e interoperável, múltiplas fontes de energia de diferentes portes e tecnologias.

6. Redução do impacto ambiental: diminuição das perdas no sistema e incentivo ao uso de fontes energéticas com baixo impacto ambiental, contribuindo para a sustentabilidade do setor elétrico.

7. Resposta à demanda: realização de ações remotas sobre dispositivos de consumo, possibilitando ajustes dinâmicos na demanda energética em tempo real.

8. Estímulo à competição no mercado de energia: criação de condições para o desenvolvimento de mercados competitivos, com destaque para o fortalecimento do mercado varejista e o incentivo à microgeração distribuída.

A implementação das redes inteligentes deverá ocorrer de maneira gradual e progressiva (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011), por meio da incorporação

contínua de tecnologias avançadas de automação, computação e comunicação em segmentos específicos do sistema elétrico. (PAUL et al., 2014) Essa abordagem dará origem a sub-redes locais com características típicas de *Smart Grids*, que coexistirão de forma integrada e compatível com a infraestrutura elétrica legada.

À medida que essas sub-redes se expandirem em número e em capacidade operacional, haverá uma transformação sistêmica: a rede elétrica como um todo tenderá a convergir para o novo paradigma tecnológico proposto pelas *Smart Grids*, consolidando uma estrutura mais inteligente, resiliente e interoperável (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011).

Essa transição para redes elétricas inteligentes é impulsionada por um conjunto de tecnologias habilitadoras que viabilizam a modernização da infraestrutura elétrica e a adoção de novos paradigmas operacionais. Entre essas tecnologias, destacam-se três, segundo Falcão (2010):

1. Geração Distribuída e Microgeração: Refere-se à crescente integração de fontes de energia renovável — como solar fotovoltaica e eólica — diretamente nos sistemas de distribuição. Essa tendência é fomentada por fatores ambientais, incentivos regulatórios e avanços tecnológicos, promovendo um modelo descentralizado de produção energética.

2. Infraestrutura de Medição Automatizada: Sistemas que realizam a coleta remota e automática de dados de consumo elétrico, enviando-os a centros de processamento. Essa infraestrutura, baseada em medidores inteligentes com capacidade de comunicação bidirecional, permite o monitoramento da demanda em tempo real e viabiliza estratégias de resposta à demanda por meio de preços dinâmicos e controle remoto de cargas.

3. Precificação Dinâmica: Com a implementação da comunicação bidirecional entre concessionárias e consumidores, torna-se possível adotar modelos tarifários dinâmicos, nos quais o preço da energia varia ao longo do dia. Essa prática incentiva a modulação do consumo e contribui para a redução dos custos operacionais e de expansão do sistema elétrico.

Um dos traços mais marcantes do conceito de redes elétricas inteligentes reside na redução progressiva das distinções tradicionais entre os segmentos de transmissão e distribuição de energia elétrica (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011), especialmente em áreas como serviços auxiliares, conexão e acesso à rede, além de aspectos relacionados à qualidade e à segurança do suprimento energético. Essa convergência funcional e estrutural reflete uma transformação profunda na arquitetura do sistema elétrico,

cuja lógica passa a incorporar princípios de flexibilidade, interoperabilidade e sinergia operacional entre diferentes níveis da infraestrutura.

Simultaneamente, essa tendência de descentralização e democratização do sistema — caracterizada pela multiplicação de pontos de geração, pela diversificação dos perfis de consumo e pela crescente autonomia dos usuários — impõe a necessidade de um reforço substancial dos mecanismos de controle e gerenciamento da rede. Tais mecanismos, por sua vez, não apenas asseguram a operação segura e confiável do sistema em um ambiente cada vez mais dinâmico (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011), como também constituem elementos indispensáveis para a implantação eficaz de novas abordagens tecnológicas, tais como as Usinas Virtuais e os modelos de gerenciamento energético voltados ao consumidor final.

Para que o sistema elétrico possa atender adequadamente às necessidades futuras dos consumidores, será essencial o surgimento e a consolidação de novos agentes no mercado de energia (HAMIDI; SMITH; WILSON, 2010) , os quais atuarão em diferentes pontos da cadeia de valor, introduzindo práticas inovadoras de oferta, gestão e uso de energia. Nesse contexto, torna-se imperativo assegurar o acesso à rede de forma transparente, equitativa e não discriminatória a todos os usuários, sejam eles geradores ou consumidores, promovendo a abertura e a competitividade do mercado (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011).

Entretanto, para viabilizar economicamente esse cenário de cooperação tecnológica e interatividade em larga escala, será imprescindível a estruturação de novos modelos de remuneração e fluxos de receita, que reconheçam e incentivem a participação ativa dos diversos atores na cadeia elétrica moderna, garantindo a sustentabilidade financeira da transição energética em direção a uma rede mais inteligente, resiliente e centrada no usuário (FALCÃO, 2010).

Como pode-se imaginar a tecnologia da informação e da comunicação desempenha um papel fundamental na estruturação das redes elétricas inteligentes, sendo responsável tanto pela definição das tarefas essenciais quanto pela implementação de normas técnicas que viabilizam o desenvolvimento e a operação de soluções baseadas em comunicação digital no contexto das redes do futuro (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011). A aplicação dessa tecnologia constitui um pré-requisito indispensável para a efetiva troca de dados entre os diversos agentes que compõem a cadeia de fornecimento de energia elétrica. Além disso, ela garante a operação segura, econômica e ambientalmente sustentável das redes inteligentes.

Entre os elementos-chave e componentes prioritários para o êxito dessa integração tecnológica, destacam-se a necessidade de uma infraestrutura de comunicação que seja simples, robusta, segura e flexível, de modo a possibilitar a monitoração, o gerenciamento, o controle e o despacho das operações até os níveis mais próximos do consumidor

final (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011); a definição de modelos comuns de dados e de informação que contemplem todos os blocos funcionais do sistema elétrico, assegurando assim a consistência das bases de dados; a implementação de soluções confiáveis de comunicação, consideradas essenciais para a manutenção da segurança do suprimento e para a promoção de uma interação eficiente entre os diferentes participantes do mercado; e, finalmente, a promoção de um ambiente competitivo para todos os produtos e serviços envolvidos, o qual somente poderá ser alcançado mediante a adoção de tecnologias baseadas em normas bem definidas e padronizadas que viabilizem a atuação de múltiplos fornecedores em condições equitativas.

2.2.2 Segurança cibernética em *Smart Grids*

A segurança cibernética (BARI A JIANG J, 2014) nas redes elétricas inteligentes constitui uma dimensão crítica para a viabilidade operacional e a resiliência sistêmica dessas infraestruturas, que se caracterizam por sua abrangência e conectividade. Uma rede inteligente é concebida como um sistema de grande escala que se estende desde os pontos centrais de geração de energia até os dispositivos individuais de consumo, tais como eletrodomésticos, computadores e dispositivos móveis. Essa vasta extensão do sistema, aliada à sua natureza digital e interconectada, ampliou significativamente as possibilidades de controle remoto da geração, distribuição e gestão da energia elétrica, ao mesmo tempo em que aumentou a vulnerabilidade a ameaças cibernéticas, furtos de energia, abusos operacionais e ações maliciosas direcionadas ao sistema.

Os desafios associados à garantia da segurança cibernética no contexto das redes inteligentes são múltiplos e variados, refletindo a diversidade dos componentes tecnológicos envolvidos e a heterogeneidade dos ambientes de aplicação. A ausência de medidas de segurança robustas e minuciosas pode permitir que ataques cibernéticos sofisticados sejam executados de forma silenciosa, sem detecção imediata, comprometendo assim a integridade e a estabilidade do sistema como um todo. Entre os possíveis impactos da fragilidade em segurança, incluem-se fraudes em serviços públicos, vazamento de informações confidenciais de usuários e acesso indevido a dados detalhados de consumo energético.

Os objetivos fundamentais da segurança cibernética (BARI A JIANG J, 2014), nesse contexto, podem ser agrupados em três categorias principais: a integridade, voltada para a proteção contra modificações ou destruições não autorizadas de dados, cuja violação pode gerar distorções na operação dos sistemas de energia; a confidencialidade, centrada na limitação do acesso e divulgação de informações privadas ou proprietárias apenas a entidades autorizadas; e a disponibilidade, que visa assegurar o acesso contínuo e confiável às informações e aos serviços, sendo essencial para a manutenção da entrega de energia.

Embora a integridade e a disponibilidade sejam (BARI A JIANG J, 2014) tradicionalmente vistas como os pilares da confiabilidade operacional das redes inteligentes, a

crescente integração de mecanismos de comunicação bidirecional com os usuários finais tem elevado substancialmente a importância da confidencialidade. Essa preocupação decorre da circulação intensiva de dados em tempo real entre medidores, coletores, redes de comunicação e centros de dados das distribuidoras, compondo uma malha de informações sensíveis que deve ser protegida contra acessos indevidos e usos indevidos.

A estrutura das redes inteligentes, baseada frequentemente em topologias de rede em malha, introduz um número elevado de pontos de acesso, incluindo dispositivos domésticos conectados às redes de automação residencial. Cada um desses pontos representa uma potencial porta de entrada para agentes maliciosos que, ao explorarem vulnerabilidades, podem obter acesso a softwares de controle e manipular as condições de carga, desestabilizando a rede de forma imprevisível. Ataques direcionados a qualquer parte do sistema podem rapidamente se propagar por toda a rede, dada sua natureza interligada. Um ponto crítico de exposição é a comunicação sem fio entre os medidores dos consumidores e os coletores de dados, que, se não forem protegidos adequadamente, se tornam alvos fáceis para ações intrusivas (BARI A JIANG J, 2014).

Para mitigar esses riscos, os mecanismos de segurança devem ser aplicados em múltiplas camadas — tanto físicas quanto lógicas. No plano físico, é necessário proteger os ativos do sistema contra interferências externas, adulterações, furtos, vandalismo e sabotagens. Isso inclui, por exemplo, a instalação de cercas, sistemas de videomonitoramento e alarmes. No plano lógico, a segurança está associada à proteção da informação digital. Dentre os mecanismos utilizados nesse domínio, destacam-se, segundo Bari (2014):

- **Criptografia**: A codificação dos dados desde o medidor até o centro de processamento das concessionárias é um instrumento essencial para assegurar a confidencialidade da informação e impedir interceptações não autorizadas (BARI A JIANG J, 2014). Os dispositivos que compõem o ecossistema da rede inteligente — como medidores, coletores, processadores e roteadores — devem estar habilitados para processar algoritmos de criptografia com elevado grau de segurança e eficiência.

- **Autenticação**: O processo de autenticação visa validar a identidade dos usuários ou dispositivos que se comunicam com o sistema, de forma a rejeitar conexões não autorizadas. Essa validação é fundamental para proteger os canais de comunicação entre os diversos elementos da rede, como os medidores e as interfaces com as concessionárias.

- **Controles de segurança em aplicações**: As aplicações embarcadas em medidores e outros dispositivos devem ser projetadas com boas práticas de programação, de modo a prevenir ataques por estouro de buffer ou inserção de códigos maliciosos. Técnicas como a validação rigorosa de dados são fundamentais nesse sentido.

- **Atualizações de segurança**: A aplicação de correções e atualizações regulares nos sistemas é crucial para eliminar vulnerabilidades conhecidas, garantindo que o sistema esteja sempre protegido contra ameaças emergentes.
- Remoção de softwares maliciosos: O uso disseminado de antivírus e sistemas de detecção de programas espiões em todos os componentes da rede contribui para a identificação e neutralização de ameaças.

Diversas abordagens têm sido propostas recentemente para fortalecer a segurança nas redes inteligentes. Entre elas, destaca-se a infraestrutura de chave pública, na qual uma autoridade certificadora associa chaves criptográficas públicas a identidades únicas de usuários, permitindo comunicações seguras e confiáveis. Também se destaca a anonimização de dados, que protege a privacidade do consumidor ao dissociar dados de consumo de informações pessoais nos casos em que isso não é estritamente necessário, como nos processos de balanceamento de carga em tempo real. Uma outra alternativa é a utilização de tecnologias modernas e seguras de armazenamento de informações, uma delas é a *blockchain* (MOLOLOTH; SAGUNA; ÅHLUND, 2023), que é tema central deste trabalho e será abordada em detalhes na sessão posterior.

2.3 BLOCK CHAIN

A tecnologia denominada *blockchain* teve sua primeira formalização conceitual apresentada no documento intitulado "Bitcoin: A Peer-to-Peer Electronic Cash System"(NAKAMOTO, s.d.), publicado sob o pseudônimo de Satoshi Nakamoto, no qual foram estabelecidas as bases matemáticas que fundamentam a criptomoeda bitcoin (PIERRO, 2017). Embora esse trabalho seja amplamente reconhecido por seu caráter inovador e disruptivo — especialmente no que se refere à concepção de um sistema financeiro digital descentralizado — é importante salientar que ele não foi submetido aos trâmites convencionais de revisão por pares em periódicos científicos. Ademais, a identidade do autor permanece desconhecida até os dias atuais.

Desde então, a tecnologia *blockchain* consolidou-se como o elemento estruturante de todas as criptomoedas modernas. No entanto, sua aplicabilidade transcende o domínio criptoativo e passou a ocupar um espaço relevante em setores mais tradicionais da indústria financeira, promovendo avanços substanciais em áreas como a rastreabilidade de transações, a automação de processos contratuais por meio de contratos inteligentes e a confiabilidade das operações digitais.

Um dos principais desafios que Nakamoto enfrentou (PIERRO, 2017) — e solucionou com o uso do *blockchain* — foi o da construção de confiança em sistemas distribuídos, particularmente no que diz respeito à criação de mecanismos capazes de armazenar documentos datados de forma cronológica e descentralizada, garantindo que nenhuma das

partes envolvidas pudesse adulterar os conteúdos ou manipular os registros de tempo sem que tal ação fosse detectada. Essa problemática é distinta, ainda que complementar, das questões clássicas da segurança da informação, como autenticação, integridade e irretratabilidade, as quais podem ser tratadas com o uso de assinaturas digitais. Embora tais assinaturas permitam estabelecer um vínculo verificável entre o autor e o conteúdo do documento, demonstrando que o documento não foi alterado desde a assinatura e que o signatário de fato a emitiu (ALLADI et al., 2019), elas não oferecem garantias quanto ao momento exato da assinatura — um fator crítico em transações financeiras e contratos legais, no qual a temporalidade é elemento jurídico essencial.

Tecnicamente, a unidade fundamental de uma *blockchain* consiste, em essência, em um número válido, derivado da resolução de uma equação matemática definida. O processo pelo qual tais soluções são encontradas denomina-se mineração, e confere ao descobridor o direito sobre a nova unidade criptográfica gerada. As transações subsequentes dessas unidades são registradas em um livro-razão digital, no qual cada operação é assinada digitalmente com as credenciais criptográficas do emissor, assegurando a não-repudiação das transações.

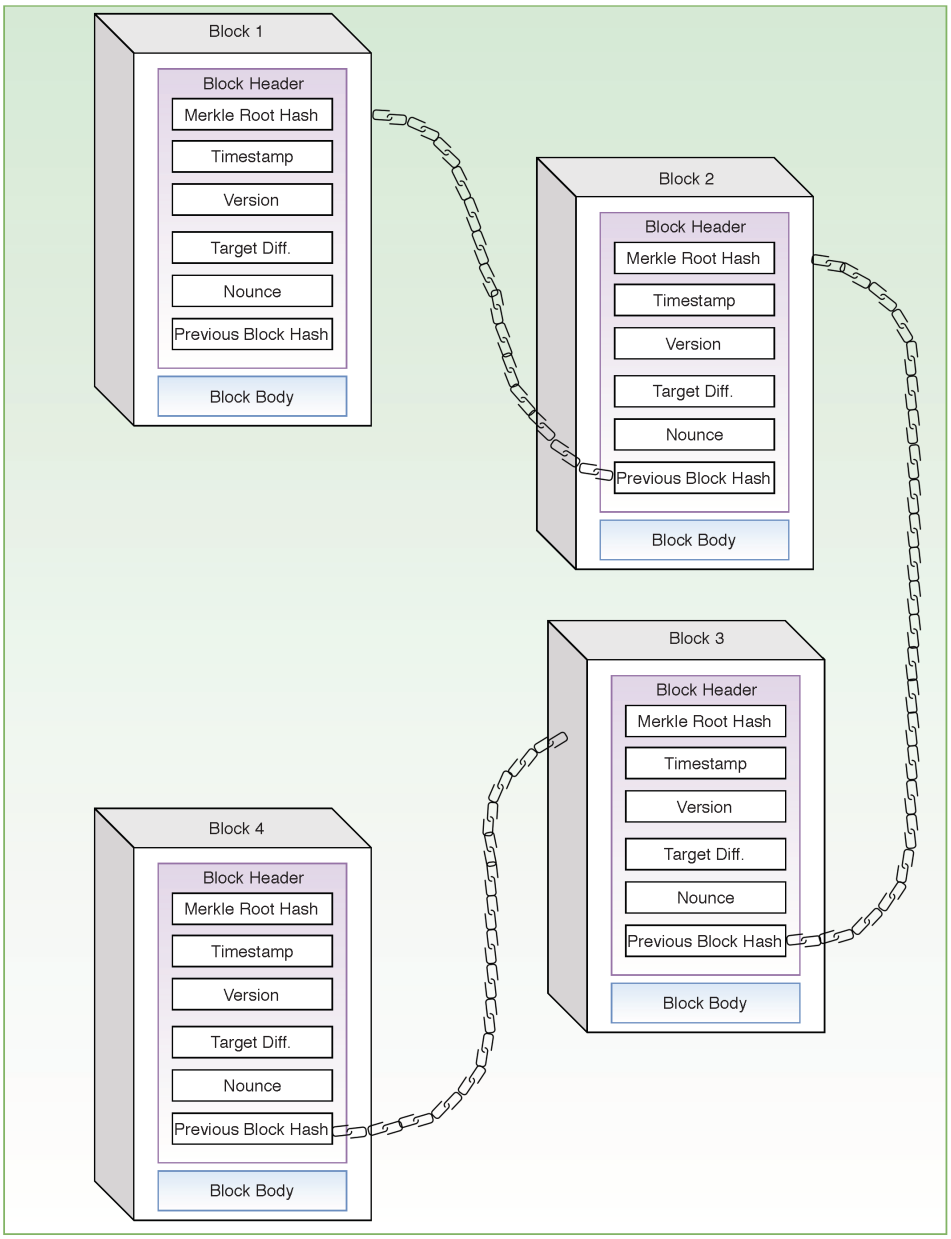
Diferentemente dos sistemas financeiros tradicionais, que dependem de entidades centralizadas para registro e verificação, o *bitcoin* e outras *criptomoedas* administradas por uma *blockchain* adotam um modelo de livro-razão distribuído, no qual cada nó participante da rede mantém uma cópia sincronizada do histórico de transações. Esse modelo elimina o risco de pontos únicos de falha, garantindo que qualquer tentativa de adulteração seja rapidamente detectada pela rede descentralizada.

As transações, nesse ambiente, são tratadas como unidades de dados contendo informações operacionais e carimbo temporal, podendo ser representadas por números ou cadeias alfanuméricas (PIERRO, 2017). A *blockchain*, estruturalmente, pode ser compreendida como uma tabela com três colunas principais (figura 4): a primeira armazena um código que representa o tempo em que o bloco foi montado, chamado *timestamp*; a segunda, os dados da transação; e a terceira, o valor de *hash* (código identificador) gerado a partir da combinação entre o conteúdo da transação corrente e o *hash* da anterior. O novo *hash* é imediatamente disseminado pela rede, permitindo que qualquer tentativa de alteração em transações anteriores gere uma inconsistência detectável e rejeitada pelo sistema.

Do ponto de vista conceitual, um *hash* pode ser entendido como um resumo criptográfico da informação original, de modo que não é possível reconstruí-la a partir do valor resultante. As funções de *hash* utilizadas nesse processo devem satisfazer dois requisitos essenciais: possuir domínios e codomínios extensos e apresentar elevada resistência a colisões, ou seja, minimizar a probabilidade de dois conteúdos distintos produzirem o mesmo *hash*.

Cada novo bloco é adicionado à cadeia (Figura 5) por meio do processo no qual os nós mineradores utilizam algoritmos de consenso para validar as transações (AHRAM et al., 2017). A integridade dessa cadeia é garantida pelo encadeamento entre blocos sucessivos, vinculados por suas respectivas funções de *hash*. Qualquer tentativa de alteração em um bloco antigo exigiria o reprocessamento de todos os blocos subsequentes, além da superação da capacidade computacional de toda a rede honesta — o que torna esse tipo de fraude economicamente inviável.

Figura 4 – Diagrama detalhando o funcionamento de uma *blockchain*

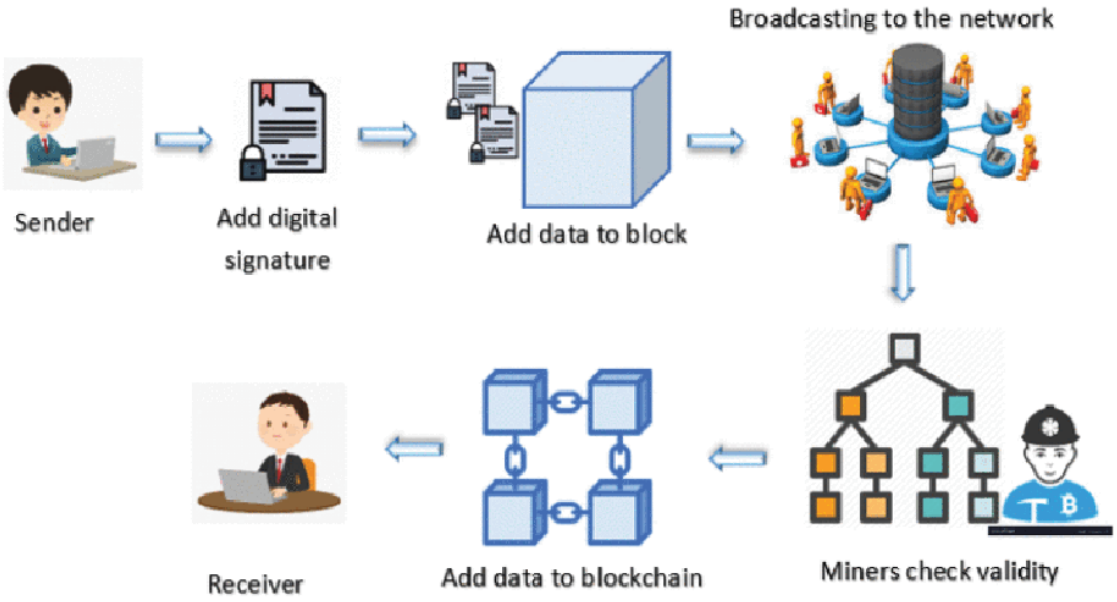


Fonte: Alladi, 2019

As tecnologias baseadas em *blockchain* podem ser classificadas, de forma geral, em dois tipos principais: *blockchains* públicas e *blockchains* privadas (ALLADI et al.,

2019). As *blockchains* públicas se caracterizam por sua natureza totalmente descentralizada, permitindo que qualquer usuário insira e visualize informações sem a necessidade de autorização prévia. Toda a estrutura é acessível, reforçando os princípios de transparência, abertura e imutabilidade. Em contraste, as *blockchains* privadas são restritas a usuários previamente autorizados, que devem garantir sua autenticação e operar sob mecanismos rigorosos de controle de acesso. Esse modelo é frequentemente adotado por organizações que requerem governança centralizada, mantendo, contudo, os benefícios da criptografia e rastreabilidade proporcionados pela tecnologia *blockchain*.

Figura 5 – Exemplo de transação utilizando *blockchain*



Fonte: Sunny, 2022

2.3.1 Aplicações de *Blockchains*

A tecnologia *blockchain* tem sido amplamente explorada em diversos setores ao longo dos últimos anos, consolidando-se como uma ferramenta estratégica para aprimorar a transparência, a segurança e a descentralização em sistemas digitais complexos. A seguir, são apresentadas algumas das principais áreas de aplicação da *blockchain* no presente e perspectivas para o futuro, com destaque para os setores de transporte inteligente e Internet das Coisas (SUNNY et al., 2022).

2.3.1.1 Blockchains em Sistemas de Transporte

O avanço dos sistemas de transporte inteligente está intrinsecamente ligado à incorporação de inovações tecnológicas voltadas à automação e à eficiência operacional. A utilização de arquiteturas baseadas em *blockchain* nesses sistemas tem demonstrado um

grande potencial para promover economia de recursos humanos, financeiros e logísticos, ao mesmo tempo em que oferece melhorias substanciais nos processos de gestão de tráfego e mobilidade urbana. Uma das propostas notáveis nessa área foi apresentada por Humayun et al., que conceberam uma estrutura chamada *Smart Transportation and Logistic Framework*, a qual integra *blockchain* e Internet das Coisas para otimizar operações logísticas e de mobilidade. Nos sistemas veiculares tradicionais, a comunicação entre veículos e infraestrutura é vulnerável a violações de privacidade, podendo expor informações sensíveis como a identidade do condutor, o tipo de veículo e sua localização. Soluções anteriores baseadas em pseudônimos apresentavam custos elevados e dificuldades na distribuição de certificados. A descentralização e a transparência oferecidas pela blockchain surgem como alternativas promissoras para superar essas limitações (SUNNY et al., 2022).

Sobre aplicações de *blockchains* em sistemas de transporte, destacam-se (SUNNY et al., 2022):

1. Redução de Tráfego Soluções baseadas em *blockchain* vêm sendo aplicadas para permitir o compartilhamento seguro de informações de tráfego em tempo real. Em um ITS proposto por Li et al., o uso da *blockchain* visa estabelecer uma infraestrutura de confiança para validar a autenticidade dos dados compartilhados entre os usuários, contribuindo para a sugestão de rotas mais eficientes e a mitigação de congestionamentos.

2. Pagamentos Inteligentes A tecnologia *blockchain* também viabiliza a gestão descentralizada de pagamentos associados ao transporte, como aquisição de passagens para ônibus, trens leves, taxas de estacionamento e registros veiculares. O uso de contratos inteligentes permite transações seguras e automáticas, reduzindo intermediários e proporcionando maior comodidade para motoristas e passageiros.

3. Troca de Energia No contexto de veículos elétricos e aeronaves não tripuladas, a necessidade de um sistema seguro e eficiente para troca de energia é crucial. Um modelo baseado em *blockchain* foi proposto para validar pedidos de carregamento de energia por meio de um mecanismo delegativo, no qual nós mineradores verificam a integridade das solicitações. Um controlador de *backbone* definido por software também integra o sistema para otimizar o gerenciamento de energia.

4. Distribuição de Dados e Gerenciamento de Chaves A gestão de grandes volumes de dados em redes veiculares exige soluções seguras e escaláveis. A blockchain facilita o gerenciamento dinâmico de chaves de segurança em ambientes altamente distribuídos, eliminando a dependência de autoridades centrais e permitindo que o processo de autenticação ocorra de forma autônoma. Propostas recentes têm demonstrado a eficácia de algoritmos de gerenciamento de chaves baseados em blockchain para melhorar o desempenho e a resiliência dos sistemas ITS.

2.3.1.2 Blockchain na Internet das Coisas

A união entre a tecnologia blockchain e a Internet das Coisas (Figura 7) é considerada uma das mais promissoras da atualidade (ALLADI et al., 2019). Cada vez mais presentes no nosso cotidiano, os dispositivos IoT (como sensores, câmeras e medidores) são usados em cidades inteligentes, hospitais, escolas e até em redes sociais. Como eles geram muitos dados e compartilham informações pela internet, é essencial garantir que tudo isso aconteça com segurança e confiança.

Nas redes IoT, cada aparelho funciona como um pequeno participante da rede, com uma espécie de "identidade digital"formada por duas chaves: uma pública e uma privada. Quando o dispositivo começa a operar, ele registra sua chave pública em uma *blockchain*. Depois, ao transmitir dados, usa a chave privada para assinar digitalmente a informação, que é então verificada por outro dispositivo da rede. Esse processo garante que os dados venham realmente do dispositivo certo e não sejam alterados no caminho (SUNNY et al., 2022).

Algumas aplicações de *blockchains* em internet das coisas são (SUNNY et al., 2022):

1. **Cidades Inteligentes:** São compostas por vários dispositivos IoT que ajudam na gestão urbana, como semáforos, câmeras de segurança ou sensores ambientais. Para evitar que um erro num único dispositivo comprometa tudo, pesquisadores criaram sistemas com *blockchain* privada.

2. **Indústria:** A combinação entre IoT e *blockchain* tem tornado as fábricas mais eficientes. Com sensores mais baratos, é possível monitorar tudo em tempo real. A *blockchain* garante que essas informações sejam armazenadas de forma segura e confiável entre várias empresas.

3. **Logística e Transporte:** Esses setores enfrentam muitos problemas como atrasos e erros humanos. A *blockchain* ajuda a digitalizar processos e a rastrear componentes com precisão. Já o IoT fornece os dados, como localização e status dos produtos. Um exemplo é o uso de contratos inteligentes que automatizam entregas e seleção de fornecedores.

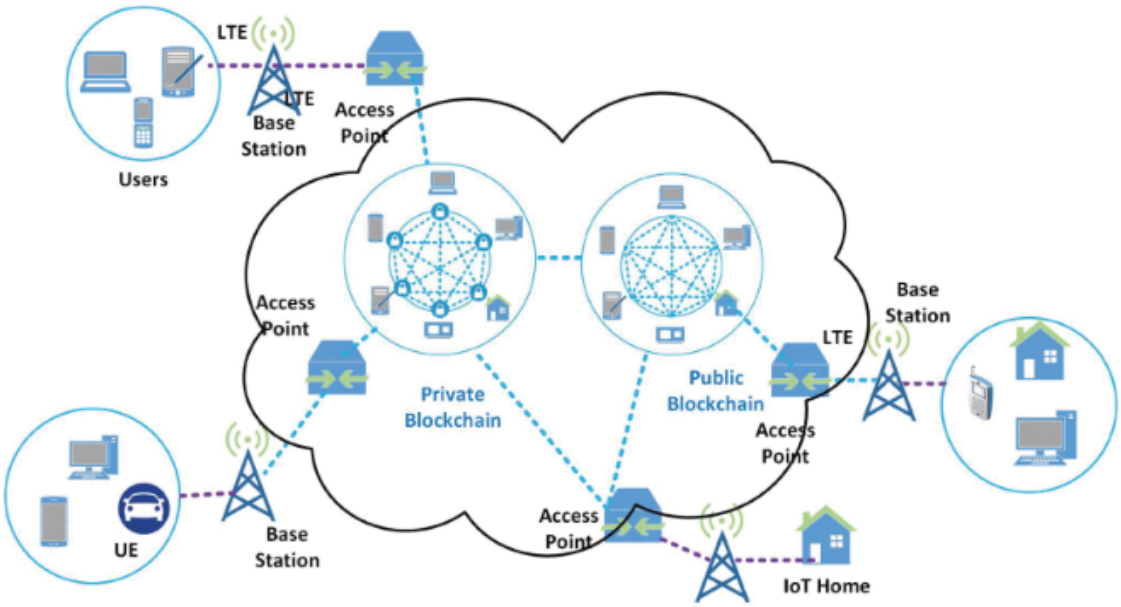
4. **Veículos Conectados:** Carros que se comunicam pela internet geram muitos dados — como manutenção, combustível ou localização. Usar *blockchain* permite registrar esses dados de forma transparente, criando confiança entre fabricantes e usuários.

5. **Confiança nos Dados:** Para garantir que os dados gerados por IoT sejam realmente confiáveis, foi criada uma arquitetura de três camadas. Primeiro, os dados são protegidos por códigos especiais (*hash*). Depois, são registrados na *blockchain*. Por fim, são processados por softwares que geram serviços para quem usa.

6. **Alimentos:** Para saber se o que estamos comendo é seguro, sistemas com IoT e *blockchain* podem acompanhar tudo — desde o cultivo até o transporte e armazenamento. Na Nova Zelândia, por exemplo, consumidores já usam essas tecnologias para verificar a origem e qualidade dos alimentos.

7. **Contratos Inteligentes:** São regras automáticas que podem acionar ações quando certas condições são cumpridas. Em redes IoT, um fabricante pode, por exemplo, atualizar aparelhos remotamente usando um contrato inteligente. Isso ajuda a manter o controle e a organização dos dispositivos conectados em setores como energia elétrica.

Figura 6 – Exemplo de arquitetura para ampla aplicação de *blockchain*



Fonte: Alladi, 2019

Essa lista de itens é apenas um pequeno resumo de todas as possíveis aplicações para a tecnologia, Farhana Akter Sunny detalha muitas outras aplicações no artigo *A Systematic Review of Applications* de 2022.

A aplicação chave para este trabalho está no uso de *blockchains* no contexto de *smart grids*. Algumas das aplicações se assemelham ao que já foi citado para internet das coisas, mas a união entre os dois conceitos será abordada na próxima sessão.

2.4 *BLOCKCHAINS* EM *SMART GRIDS*

A crescente complexidade dos sistemas elétricos modernos exige soluções inovadoras que garantam segurança, transparência, escalabilidade e interoperabilidade. Nesse cenário, a tecnologia *blockchain* tem-se destacado como uma ferramenta promissora, especialmente no contexto de *smart grids* — redes elétricas inteligentes que integram recursos de geração distribuída, consumo automatizado e comunicação em tempo real (AHRAM et al., 2017).

As redes inteligentes buscam operar de forma mais eficiente e resiliente, incorporando tecnologias avançadas de sensoriamento, monitoramento e análise de dados. A inclusão da *blockchain* nesse ecossistema permite a criação de registros imutáveis e descentralizados, o que fortalece os mecanismos de confiança entre os diversos agentes da rede, desde concessionárias e consumidores até microgeradores e gestores energéticos. Neste capítulo, exploramos as principais aplicações da tecnologia *blockchain* em *smart grids*, com ênfase em seus impactos técnicos, operacionais e estratégicos (ALLADI et al., 2019).

Alladi (2019) apresenta uma representação esquemática dos principais casos de uso da tecnologia *blockchain* em *smart grids*, focando em seis principais itens:

- Negociação de energia entre pares (*peer-to-peer*)
- Negociação de energia em veículos elétricos
- Técnicas de preservação de segurança e privacidade
- Manutenção segura de equipamentos
- Geração e distribuição de energia
- Gestão confiável de infraestrutura

Na sessão seguinte, serão detalhados alguns desses itens.

2.4.1 Casos de uso de *blockchains* em *smart grids*

2.4.1.1 Compra de energia peer-to-peer

Um dos principais entraves dos sistemas elétricos convencionais é a ausência de mecanismos robustos de segurança nas transações energéticas, agravada pela presença de intermediários e entidades centralizadoras que atuam como terceiros em todas as etapas

da negociação. Esse modelo hierárquico, característico da infraestrutura tradicional de comercialização de energia, impõe custos operacionais elevados e limita significativamente a eficiência dos processos de distribuição, monitoramento e tomada de decisão (ALLADI et al., 2019).

Em contrapartida, a adoção de uma infraestrutura de negociação baseada na tecnologia *blockchain* oferece uma alternativa descentralizada, transparente e segura para o comércio de energia, viabilizando a negociação direta entre pares — conhecida como comércio *Peer-to-Peer*. Nesse novo paradigma, consumidores (que apenas consomem energia) e prosumidores (que geram e consomem energia) podem transacionar energia entre si sem depender de intermediários centralizados (ALLADI et al., 2019).

A principal vantagem dessa abordagem está no fortalecimento da privacidade das identidades dos participantes e na segurança criptográfica das transações, que são registradas de forma imutável na *blockchain*. Esse modelo também promove maior autonomia para os usuários, que passam a controlar diretamente seus recursos energéticos e decisões de consumo. O uso de contratos inteligentes permite automatizar as condições da negociação — como volumes, preços e prazos —, assegurando que os acordos sejam executados de maneira precisa e auditável .

O comércio de energia peer to peer por meio de *blockchain* possui inúmeras aplicações, sendo especialmente relevante no contexto da Internet das Coisas Industrial (IIoT). Nesse cenário, dispositivos conectados e sensores podem atuar como agentes autônomos, gerenciando fluxos de energia, detectando padrões de consumo e interagindo com microredes locais (microgrids). As microgrids são estruturas energéticas descentralizadas que operam de forma autônoma e colaborativa, permitindo o uso sustentável da energia gerada em comunidades, edifícios comerciais ou instalações industriais (MOLOLOTH; SAGUNA; ÅHLUND, 2023).

Além de seu valor operacional, essa transformação estrutural também tem implicações econômicas estratégicas. Organizações e governos têm anunciado planos robustos de investimento em mercados energéticos locais, buscando acelerar a transição para redes inteligentes baseadas em *blockchain*. Um dos objetivos centrais é fomentar ambientes energéticos mais democráticos, eficientes e ecológicos, em que cada usuário desempenha um papel ativo na geração, consumo e comercialização de energia (ALLADI et al., 2019).

Com base nas pesquisas mais avançadas sobre infraestrutura de negociação de energia *Peer-to-Peer* utilizando tecnologia *blockchain*, tem-se delineado uma arquitetura típica para sistemas transacionais que operam nesse modelo. Essa arquitetura pode ser representada esquematicamente e é estruturada segundo modelos de referência que priorizam segurança, escalabilidade e descentralização (ALLADI et al., 2019).

A adoção desse sistema arquitetura apresenta vantagens significativas para o desen-

volvimento de redes energéticas inteligentes (MOLOLOTH; SAGUNA; ÅHLUND, 2023):

- Maior segurança e imutabilidade dos registros
- Redução de intermediários e de custos operacionais
- Escalabilidade com controle sobre o consumo energético
- Estímulo à geração distribuída e à participação de prosumidores
- Facilidade de integração com tecnologias IIoT e algoritmos de automação

Essa abordagem viabiliza mercados locais de energia em comunidades, indústrias e regiões conectadas, promovendo transparência, eficiência energética e sustentabilidade. A estrutura de um bloco na *blockchain* desse contexto é composta por campos como:

- Identificador do Bloco: código único que diferencia cada bloco na cadeia.
- Cabeçalho: campo criptografado com algoritmos *hash* seguros (por exemplo, SHA256), contendo metadados essenciais para validação.
- Horário de Registro: marca temporal indicando o momento de inserção do bloco na rede.
- Transações: lista das operações registradas, cada uma associada a metadados específicos.

Para essa estrutura em um bloco é necessário o desenvolvimento de um **contrato inteligente**, uma forma de especificação do formato do bloco para a garantia que todas as informações úteis para essa transação específica (no caso, compra de energia) estão presentes (MOLOLOTH; SAGUNA; ÅHLUND, 2023).

Os contratos inteligentes são códigos autoexecutáveis que contêm cláusulas de negociação. Em *smart grids*, eles são implementados nos medidores inteligentes conectados à *blockchain*, assegurando que apenas dados autênticos sejam transmitidos entre dispositivos e nós supervisores.

2.4.1.2 Técnicas de Segurança e Preservação de Privacidade em Smart Grids com Blockchain

Nos sistemas de redes elétricas inteligentes, os medidores digitais instalados nas residências desempenham um papel central na coleta de dados em tempo real sobre o consumo de energia elétrica. Esses dados são utilizados pelas concessionárias para diversos fins, como otimização da distribuição, tarifação dinâmica e análise de demanda. No entanto, a coleta contínua desses registros pode representar uma vulnerabilidade à

privacidade dos usuários, uma vez que o perfil de consumo energético pode ser analisado para inferir comportamentos pessoais, como horários em que o imóvel está ocupado ou padrões de rotina da família (ALLADI et al., 2019).

Para lidar com essas preocupações, arquiteturas que integram *blockchain* têm sido exploradas como mecanismos eficazes de segurança e preservação de privacidade. Em modelos mais avançados, os usuários da rede são organizados em grupos distintos, e cada grupo opera com sua própria instância de *blockchain*. Essa abordagem distribui o armazenamento e processamento dos dados de forma controlada, permitindo que informações sejam registradas sem revelar diretamente a identidade dos envolvidos.

Uma das soluções utilizadas para garantir que apenas usuários legítimos participem das transações é o uso de filtros probabilísticos que permitem verificar rapidamente se determinada identidade está autorizada a operar na rede. Esses filtros são eficientes para autenticação sem expor dados pessoais, funcionando como uma barreira leve e eficaz contra invasores (MOLOLOTH; SAGUNA; ÅHLUND, 2023).

Além disso, para reforçar o anonimato, os sistemas empregam a atribuição de pseudônimos, permitindo que os usuários interajam com o sistema sem utilizar seu nome real ou qualquer identificador sensível. Embora a estrutura da *blockchain*, por si só, não seja suficiente para garantir privacidade absoluta — uma vez que todos os registros são públicos e imutáveis — é possível incorporar técnicas criptográficas avançadas que expandem suas capacidades em relação à confidencialidade dos dados (ALLADI et al., 2019).

Entre essas técnicas, destacam-se:

- Prova de conhecimento nulo: mecanismo pelo qual um agente pode demonstrar que possui determinada informação ou que realizou certa operação sem revelar seu conteúdo. Isso permite validar transações ou acessos sem expor dados privados.
- Assinaturas digitais baseadas em curvas elípticas: oferecem autenticação segura e eficiente, permitindo verificar a autoria de uma transação com alto grau de proteção contra falsificações.
- Assinaturas em anel com vínculo verificável: técnica que permite que um grupo de usuários assine uma transação de forma que seja possível comprovar que um dos membros a realizou, mas sem revelar qual deles foi especificamente. Isso adiciona uma camada extra de anonimato à operação.

Essas técnicas podem ser combinadas com os mecanismos da *blockchain* para formar uma arquitetura robusta que protege simultaneamente a integridade dos dados e a privacidade dos usuários. A integração entre criptografia, pseudonimização e descentralização gera um sistema resiliente a ataques, confiável para os participantes e compatível

com os requisitos éticos e legais que regem a gestão de dados pessoais em ambientes digitais (ALLADI et al., 2019).

A implementação de mecanismos de segurança e preservação de privacidade em redes elétricas inteligentes não é apenas teórica — diversas iniciativas já demonstram o potencial dessa abordagem em ambientes reais, tanto urbanos quanto industriais. A seguir, são explorados alguns exemplos práticos que ilustram como essas tecnologias são aplicadas para garantir anonimato, segurança e confiabilidade dos dados em sistemas energéticos distribuídos.

Cidades Inteligentes com Tarifação Sensível ao Perfil:Em projetos-piloto de cidades inteligentes, como os desenvolvidos em países europeus e na Ásia, *smart meters* foram integrados com sistemas *blockchain* capazes de registrar, de forma segura, o consumo energético por residência. Nesses casos, os usuários recebem perfis energéticos anônimos associados a pseudônimos, evitando que seus hábitos sejam identificáveis por autoridades ou terceiros.

Os contratos inteligentes são programados para processar a tarifação com base em horários de pico, níveis de consumo e participação em programas de energia renovável. O sistema apenas reconhece o pseudônimo do usuário, mantendo os dados pessoais fora do alcance de observadores externos — inclusive das próprias concessionárias, que recebem dados técnicos, mas não os perfis individuais completos.

Indústrias com Compartilhamento Seguro de Dados Operacionais: No setor industrial, algumas fábricas conectadas a redes inteligentes começaram a implementar *blockchain* como ferramenta de rastreabilidade e sigilo operacional. Equipamentos como inversores e transformadores são monitorados por sensores conectados via rede, e os dados são registrados em blocos imutáveis. Para preservar a confidencialidade de processos produtivos, as empresas utilizam mecanismos de assinatura criptográfica e provas de conhecimento nulo, que permitem verificar a autenticidade dos dados sem revelar o conteúdo exato.

Isso protege informações críticas como padrões de produção, horários de operação e consumo de máquinas específicas — dados que poderiam ser explorados por concorrentes, se acessíveis.

Redes Comunitárias com Agregadores Privados: Em comunidades energéticas que operam micro-redes autônomas, *blockchain* tem sido empregado para registrar transações entre vizinhos de forma anônima. Cada prosumidor ou consumidor possui um endereço digital vinculado a pseudônimos, e as negociações são feitas por contratos inteligentes que validam os dados sem expor identidade.

Além disso, o sistema permite que agregadores — entidades que intermediam a gestão energética local — tenham acesso apenas aos metadados essenciais para equilíbrio da rede, enquanto os dados pessoais são protegidos por criptografia de múltiplas camadas e autenticação distribuída. Isso gera confiança entre os participantes, incentivando o uso compartilhado e sustentável dos recursos energéticos(ALLADI et al., 2019).

2.4.1.3 Manutenção preditiva e segurança de equipamentos em smart grids

A eficiência operacional (MOLOLOTH; SAGUNA; ÅHLUND, 2023) e a confiabilidade dos sistemas elétricos modernos dependem diretamente da qualidade das práticas de manutenção e do monitoramento contínuo dos ativos de rede. Em especial nas redes inteligentes, compostas por elementos tecnológicos distribuídos como subestações automatizadas, sensores avançados, controladores remotos e medidores digitais instalados em unidades consumidoras, a gestão da saúde dos equipamentos torna-se uma atividade estratégica (ALLADI et al., 2019).

Historicamente, os métodos convencionais de diagnóstico de falhas exigiam deslocamento de técnicos até os locais afetados, o que acarretava elevados custos operacionais e logísticos, além de aumentar o tempo de resposta diante de interrupções. O sucesso do processo dependia fortemente da disponibilidade de mão de obra especializada, da localização geográfica do problema e da capacidade de acesso às instalações. A insatisfação do cliente diante de períodos prolongados de espera e serviço técnico limitado reforçou a necessidade de transformação digital nos procedimentos de manutenção elétrica.

Nesse contexto, os sistemas de *smart grid* introduzem o conceito de manutenção inteligente, apoiada por dispositivos interconectados que permitem o acompanhamento remoto e em tempo real da condição dos equipamentos. Os sensores distribuídos coletam variáveis elétricas, térmicas e mecânicas de forma contínua, e algoritmos de análise são aplicados para prever falhas, estimar ciclos de vida dos componentes e sugerir intervenções antes que interrupções ocorram — abordagem conhecida como manutenção preditiva.

Além de reduzir o tempo de inatividade, (MOLOLOTH; SAGUNA; ÅHLUND, 2023) esse modelo permite que as ações sejam realizadas de forma automática e localizada, com mínima intervenção humana. Os dados gerados são registrados em plataformas seguras, podendo ser integrados à infraestrutura *blockchain* para garantir a rastreabilidade, a autenticidade e a imutabilidade dos registros de manutenção (ALLADI et al., 2019).

A diversidade dos equipamentos que compõem uma rede inteligente — incluindo relés, transformadores, inversores, medidores bidirecionais e sistemas de controle — exige que as soluções de manutenção sejam flexíveis e adaptáveis. Tecnologias como aprendizado de máquina, análise de séries temporais e algoritmos heurísticos são utilizadas para correlacionar padrões de desgaste, prever pontos de falha e sugerir substituições ou

atualizações de *firmware*.

Assim, (ALLADI et al., 2019) o gerenciamento de ativos nas *smart grids* evolui de um modelo reativo para uma abordagem proativa e digital, resultando em:

- Diminuição significativa dos custos operacionais
- Aumento da confiabilidade energética
- Redução da necessidade de visitas técnicas presenciais
- Maior integração com sistemas de resposta à demanda e de tarifação dinâmica

Essa evolução também cria oportunidades para desenvolvimento de *dashboards* operacionais, nos quais operadores podem visualizar em tempo real o estado da rede, programar intervenções e simular cenários futuros com base nos históricos registrados. Com o avanço contínuo da digitalização no setor elétrico, espera-se que os sistemas de manutenção inteligente baseados em dados e *blockchain* se tornem um componente essencial da infraestrutura energética do futuro.

2.4.2 Estudo e Comparação de Estratégias para Implementação de *Blockchain* em *Smart Grids*

É necessário investigar diferentes estratégias para estruturar uma infraestrutura *blockchain* capaz de atender às necessidades específicas do ambiente de *smart grid*. A escolha da arquitetura blockchain influencia diretamente aspectos como escalabilidade, segurança, privacidade, consumo energético e flexibilidade de implementação.

Dentre as alternativas estudadas, destacam-se quatro abordagens principais, cada uma com características distintas e potencial de aplicação no setor energético.

2.4.2.1 Implementação com blockchain Ethereum Local usando Ganache

A implementação de uma **rede Ethereum privada utilizando o ambiente de desenvolvimento Ganache**. Essa escolha se justifica pela maturidade da plataforma Ethereum, a ampla documentação disponível, e seu suporte nativo a contratos inteligentes (GANACHE, 2025).

O Ganache é uma ferramenta local que executa uma blockchain Ethereum permitindo (ETHEREUM, 2025; GANACHE, 2025):

- Criação de contas com saldo inicial de ether;
- Implantação e testes de contratos inteligentes escritos em Solidity;

- Visualização detalhada de transações, blocos e eventos;
- Comunicação com front-end via bibliotecas como Web3.js;
- Execução rápida e controle sobre o estado da rede.

Essa abordagem pode ser utilizada para modelar e validar o comportamento da negociação peer-to-peer de energia, criando tokens representativos de energia e simulações de transações entre usuários fictícios. O uso de Ganache permite realizar testes iterativos sem custos computacionais elevados, preparando a estrutura para posterior migração a ambientes reais.

2.4.2.2 Desenvolvimento de blockchain Própria com Estrutura Modular

Uma segunda abordagem consiste na construção de uma blockchain própria. Essa opção permite definir regras específicas para os blocos, algoritmos de consenso otimizados para o ambiente energético, identidade digital personalizada e controle absoluto sobre os nós da rede (HELLWIG; KARLIC; HUCHZERMEIER, 2020).

Essa abordagem é mais complexa e exige maior esforço técnico, mas oferece a liberdade para criar uma rede *blockchain* totalmente adaptada aos requisitos do sistema elétrico local.

2.4.2.3 Uso de Plataformas blockchain Permissionadas

Outra alternativa é utilizar uma plataforma **permissionada**, como o Hyperledger Fabric, que oferece um ecossistema modular voltado para ambientes corporativos e industriais. Essa arquitetura é ideal para redes energéticas, pois permite (HYPERLEDGER, 2025):

- Definir papéis e permissões entre participantes (prosumidores, concessionárias, operadores);
- Executar contratos inteligentes encapsulados em *chaincode*;
- Garantir confidencialidade com canais privados entre subconjuntos de nós;
- Alta escalabilidade e suporte a múltiplos tipos de ativos energéticos digitais.

No modelo permissionado, apenas entidades autorizadas podem participar do processo de consenso, o que reduz o consumo de recursos computacionais e melhora a performance geral. Essa configuração é especialmente útil em projetos comunitários, cooperativas energéticas ou microrredes geridas por consórcios públicos e privados.

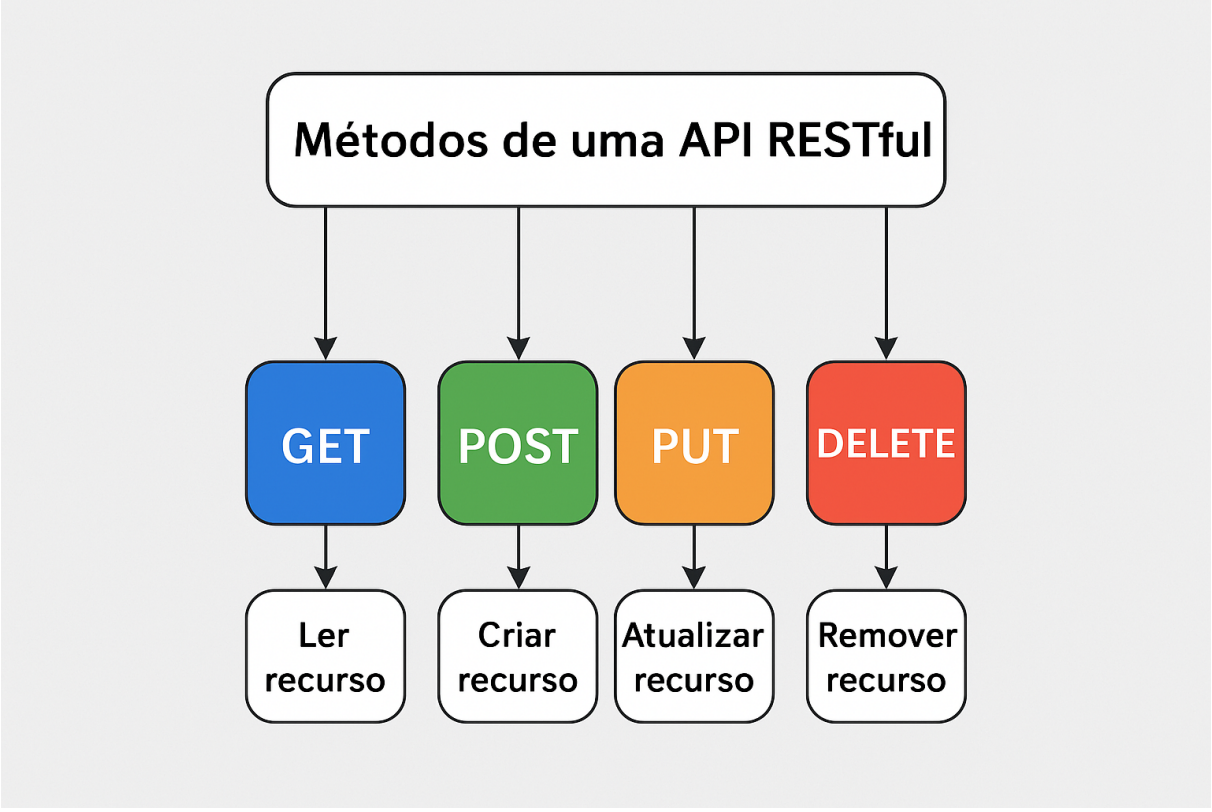
2.5 *APPLICATION PROGRAMMING INTERFACE* (API)

As Interfaces de Programação de Aplicações (MASSÉ, 2012), ou simplesmente APIs (Application Programming Interfaces), representam um conjunto de regras, padrões e protocolos que permitem a comunicação entre diferentes componentes de software. Por meio de APIs, sistemas diversos podem compartilhar funcionalidades ou trocar informações de forma estruturada e segura.

Um exemplo prático pode ser observado na integração entre uma aplicação móvel de previsão do tempo e o sistema central de um instituto meteorológico. A aplicação utiliza chamadas de API para requisitar dados atualizados, como temperatura, umidade e previsão semanal, que são processados pelo sistema meteorológico e devolvidos ao aplicativo — resultando em informações exibidas diretamente ao usuário (MASSÉ, 2012).

A comunicação entre APIs ocorre, em geral, segundo o modelo cliente-servidor. A aplicação que inicia uma requisição atua como cliente, enquanto o sistema que recebe essa solicitação e retorna uma resposta desempenha o papel de servidor. Esse modelo é altamente utilizado em sistemas distribuídos e aplicações web, como o exemplo citado anteriormente sobre previsão do tempo (AMAZON, 2024).

Figura 7 – Métodos de uma API Restful



Fonte: Elaborada pelo autor

2.5.1 Modelos de Implementação de APIs

As APIs podem ser implementadas por meio de diferentes arquiteturas, cada uma com suas características históricas e técnicas (AMAZON, 2024):

APIs SOAP (Simple Object Access Protocol): utilizam o padrão XML para envio e recebimento de dados. Apesar de oferecer robustez e segurança, são menos flexíveis e mais verbosas, sendo amplamente utilizadas em sistemas legados.

APIs RPC (Remote Procedure Call): operam com base em chamadas de procedimentos remotos, permitindo que o cliente execute funções diretamente no servidor, com retorno imediato dos resultados. São eficientes em sistemas de baixo nível ou aplicações internas.

APIs WebSocket: proporcionam comunicação bidirecional contínua entre cliente e servidor, utilizando objetos JSON. São particularmente úteis em aplicações que exigem atualização em tempo real, como chats, jogos e plataformas de monitoramento ao vivo.

APIs REST: atualmente são as mais populares e versáteis, adotadas amplamente em sistemas web e mobile. Permitem que dados sejam acessados, modificados ou excluídos por meio de comandos padronizados, como GET, POST, PUT e DELETE. São simples, eficientes e compatíveis com a estrutura do protocolo HTTP.

2.5.2 REST: Transferência Representacional de Estado

O modelo REST (Representational State Transfer) define uma arquitetura de comunicação que prescinde do armazenamento de estado por parte do servidor. Ou seja, cada requisição do cliente é independente, sem necessidade de memória contextual entre uma chamada e outra. Isso contribui para escalabilidade, desempenho e simplicidade na implementação (AMAZON, 2024).

As APIs REST operam com URLs como identificadores de recursos e utilizam respostas em formatos como JSON, promovendo interoperabilidade entre sistemas diversos.

As integrações de API consistem em mecanismos que automatizam o fluxo de dados entre sistemas distintos, permitindo que atualizações ocorram de forma transparente e eficiente. Exemplos cotidianos incluem:

- Sincronização de fotos entre o smartphone e a nuvem.
- Atualização automática de data e hora no notebook ao mudar de fuso horário.
- Integração de sistemas empresariais que conectam softwares de gestão, faturamento, estoque e atendimento.

Essas integrações tornam os ambientes computacionais mais conectados e responsivos, permitindo que sistemas interajam de maneira autônoma, dinâmica e segura — habilitando uma infinidade de aplicações modernas.

2.5.3 Benefícios das APIs REST

As **APIs REST** se consolidaram como padrão de comunicação entre sistemas web e móveis devido à sua simplicidade, flexibilidade e eficiência. Sua arquitetura orientada a recursos e baseada no protocolo HTTP permite que aplicações interajam de forma transparente e escalável. A seguir, são apresentados os principais benefícios das APIs REST para o desenvolvimento e integração de sistemas (MASSÉ, 2012).

Integração: APIs REST possibilitam a integração de novas aplicações com sistemas de software já existentes. Essa capacidade reduz significativamente o tempo e o esforço de desenvolvimento, pois funcionalidades previamente implementadas podem ser reutilizadas por meio de chamadas de API. Dessa forma, desenvolvedores não precisam escrever módulos do zero, podendo se concentrar nas partes inovadoras da aplicação. Essa característica é essencial em ambientes corporativos nos quais diversas soluções precisam trabalhar de forma sincronizada.

Inovação: A adoção de APIs REST acelera o ciclo de inovação tecnológica. Com APIs bem definidas, empresas podem adaptar rapidamente seus serviços e produtos para atender às novas demandas do mercado, modificando apenas os pontos de integração em vez de redesenhar sistemas inteiros. Isso permite o lançamento ágil de aplicações que aproveitam dados e funcionalidades existentes, favorecendo a competitividade e a inovação contínua.

Expansão: Outro benefício relevante é a facilidade de expansão para diferentes plataformas. APIs REST são compatíveis com diversos ambientes, como aplicativos web, dispositivos móveis (Android, iOS), sistemas embarcados e serviços em nuvem. Um exemplo clássico é a API de mapas, que pode ser integrada em sites, aplicativos móveis ou plataformas empresariais. Empresas podem disponibilizar acesso aos seus dados internos por meio de APIs públicas ou privadas, ampliando o alcance de seus serviços sem replicar código ou lógica de negócios.

Manutenção: A arquitetura REST promove baixo acoplamento entre os sistemas integrados. Como as APIs funcionam como *gateways* entre aplicações, cada parte pode evoluir internamente sem impactar o funcionamento da outra, desde que as interfaces contratadas permaneçam consistentes. Isso facilita a manutenção e atualização de componentes, permitindo que alterações locais não comprometam a estabilidade do sistema integrado como um todo.

2.6 CONTÊINERES E DOCKER: CONCEITOS, FUNCIONAMENTO E APLICAÇÕES

Os **contêineres** representam (DOCKER, 2024) uma abordagem moderna e altamente eficiente para a execução de aplicações em ambientes isolados e portáveis. Essa tecnologia permite empacotar uma aplicação junto com todas as suas dependências — bibliotecas, variáveis de ambiente, ferramentas e arquivos de configuração — de forma encapsulada, garantindo que o código seja executado da mesma maneira em qualquer ambiente.

2.6.1 Conceito de Contêineres

Um contêiner é uma unidade leve e portátil que compartilha o kernel do sistema operacional, mas opera em espaço isolado. Isso significa que múltiplos contêineres podem ser executados simultaneamente no mesmo host, cada um com seu próprio ambiente, sem interferência mútua. Diferentemente de máquinas virtuais (VMs), os contêineres não exigem um sistema operacional completo dentro de cada instância, o que reduz consideravelmente o tempo de inicialização e o uso de recursos (DOCKER, 2024; ANDERSON, 2015).

Contêineres são úteis em diversas situações. Eles permitem a criação de ambientes de desenvolvimento consistentes, possibilitando que os desenvolvedores testem aplicações em condições que reproduzem fielmente o ambiente de produção. Além disso, favorecem a escalabilidade de microserviços, já que aplicações modernas podem ser divididas em componentes menores — cada um rodando em seu próprio contêiner — o que facilita tanto a orquestração quanto o crescimento da solução. Por fim, contêineres também contribuem para a automação de implantação, pois podem ser integrados a pipelines de integração e entrega contínua, viabilizando atualizações rápidas e seguras.

2.6.2 Docker: Plataforma de Contêineres

O **Docker** é uma plataforma que simplifica a criação, execução e gerenciamento de contêineres. Ele fornece ferramentas e interfaces que permitem que desenvolvedores e engenheiros de infraestrutura empacotem suas aplicações com todos os requisitos em imagens prontas para execução (ANDERSON, 2015).

Segundo a documentação oficial do Docker (2024), a plataforma oferece um conjunto de recursos que simplifica o desenvolvimento, a implantação e o gerenciamento de aplicações em ambientes isolados. Um dos elementos centrais são as **imagens Docker**, que funcionam como moldes imutáveis contendo todo o sistema de arquivos necessário para rodar uma aplicação — incluindo código, bibliotecas, dependências e configurações.

O responsável por operar essas imagens é o **Docker Engine**, um mecanismo que constrói, executa e gerencia os contêineres, permitindo que eles sejam iniciados rapida-

mente e funcionem de forma eficiente. Para facilitar ainda mais esse processo, os usuários contam com ferramentas como a **Docker CLI** e o **Docker Compose**, que ajudam a definir, executar e orquestrar múltiplos contêineres ao mesmo tempo, especialmente úteis em sistemas complexos baseados em microserviços.

Além disso, a plataforma oferece o **Docker Hub**, um repositório público que funciona como uma espécie de biblioteca online, na qual os usuários podem encontrar, compartilhar e reutilizar imagens de forma colaborativa — promovendo agilidade e padronização nos projetos de software (DOCKER, 2024).

Os contêineres podem ser vinculados a volumes para persistência de dados, redes para comunicação entre serviços, e configurados com variáveis de ambiente para customização (ANDERSON, 2015).

O uso de contêineres, especialmente por meio da plataforma Docker, revoluciona o modo como aplicações são desenvolvidas, testadas e implantadas. Sua leveza, portabilidade e confiabilidade fazem dessa tecnologia um pilar da computação moderna, sendo indispensável em arquiteturas orientadas a serviços, ambientes de integração contínua e computação em nuvem (ANDERSON, 2015).

3 DESENVOLVIMENTO DA APLICAÇÃO

O método usado para a produção deste trabalho foi uma análise bibliográfica sobre as *smart grids* e a escolha de implementação de uma demanda tecnológica que garantiria um ponto de partida para a ampla utilização de energia nesse contexto. Do que foi levantado, destacam-se:

3.1 ESTUDO DE *SMART GRIDS* E MAPEAMENTO CONTEXTUAL DO SETOR ELÉTRICO NACIONAL

Como ponto de partida metodológico deste trabalho, foi conduzido um estudo aprofundado sobre o conceito de redes elétricas inteligentes (visto no capítulo anterior), com o objetivo de compreender seus fundamentos técnicos, potenciais aplicações e os principais desafios para sua implementação no contexto brasileiro. Essa etapa se mostrou essencial para dar sustentação teórica aos objetivos propostos.

A investigação sobre *smart grids* envolveu uma abordagem exploratória, voltada à identificação das características estruturais que diferenciam essas redes dos sistemas elétricos tradicionais. Foram examinados aspectos como (FALCÃO, 2010; ALLADI et al., 2019; SUNNY et al., 2022):

- O papel dos *medidores inteligentes* na coleta de dados de consumo em tempo real;
- A utilização de sensores distribuídos para monitoramento de rede e resposta automática a falhas;
- A automação de subestações, controle remoto de dispositivos e comunicação bidirecional entre concessionária e consumidor;
- A integração com fontes de energia renovável e sistemas de armazenamento distribuído;
- A possibilidade de participação ativa do consumidor, como agente produtor (prosumidor) e negociador de energia.

Tornou-se evidente que a implementação de *smart grids* representa uma oportunidade estratégica para reconfigurar o setor elétrico, promovendo **eficiência energética, equidade no acesso e inovação operacional** (FALCÃO, 2010). Este estudo preliminar também ofereceu motivação para a etapa posteriore da pesquisa: o desenvolvimento de ferramentas descentralizadas para comercialização de energia entre prosumidores e consumidores.

3.2 DETECÇÃO DAS PRINCIPAIS DEMANDAS DE DESENVOLVIMENTO PARA A APLICAÇÃO DE ENERGIA INTELIGENTE

Diante do objetivo deste trabalho de desenvolver uma aplicação prática que pudesse ajudar a integrar um modelo de *smart grid*, foi escolhido desenvolver uma forma de realizar a negociação descentralizada de energia elétrica utilizando uma *blockchain*, visto nas aplicações de *blockchain* e como uma demanda tecnológica para a implementação deste tipo de sistema energético nos capítulos anteriores (ALLADI et al., 2019; FALCÃO, 2010):

3.2.1 Negociação Descentralizada de Energia com *Blockchain*

O ponto focal levantado está na criação de uma plataforma para **compra e venda descentralizada de energia entre pares**. Nesse sistema, consumidores que produzem excedente energético (por exemplo, via painéis solares) podem negociar diretamente com outros usuários da rede sem a necessidade de intermediários.

Para garantir segurança, transparência e rastreabilidade nas transações, adota-se a tecnologia *blockchain* em um sistema que possa garantir os seguintes requisitos:

- **Confiabilidade** já que é necessário garantir que as transações são seguras e persistentes;
- **Facilidade de uso** para que usuários comuns do mercado consigo utilizar um sistema sem precisar conhecer de detalhes técnicos de implementação;
- **Registro imutável das operações**, com identificação por pseudônimos e criptografia dos dados de negociação;

Essa solução promove o surgimento de **mercados locais de energia**, incentiva a geração renovável, e redistribui o protagonismo energético ao consumidor, em alinhamento com os princípios das redes inteligentes e da sustentabilidade.

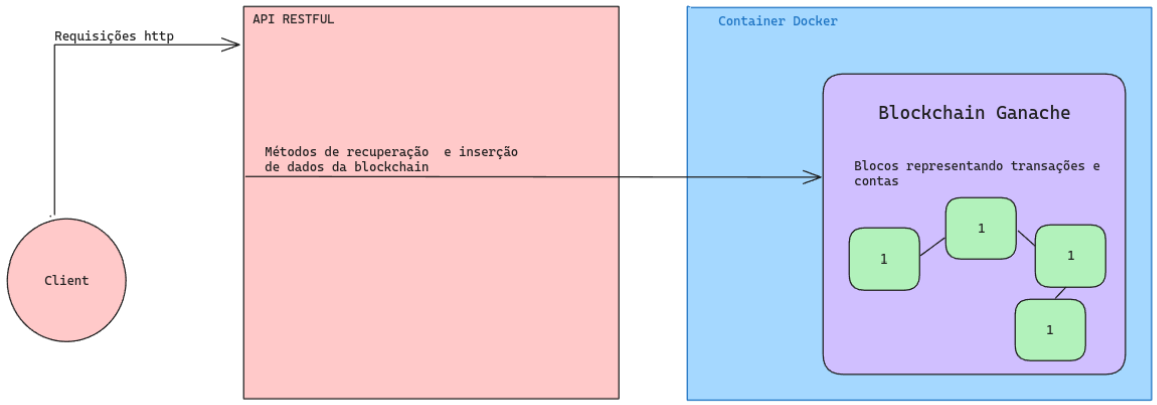
Foi escolhido trabalhar na implementação de uma maneira simples de fazer transações seguras e eficientes para o mercado de *smart grid* utilizando *blockchain*, já que o método para a implementação do mercado ainda é discutido na literatura (FALCÃO, 2010).

Considerando o estudo feito na sessão 2.4.2, foi decidida a utilização de uma *blockchain Ethereum* com *Ganache*, a solução foi considerada a melhor estratégia para testes locais e validação de conceitos, mas as demais abordagens permanecem viáveis para aplicações reais em escala regional ou nacional.

3.3 ARQUITETURA DA SOLUÇÃO IMPLEMENTADA

A escolha por uma ***API RESTful*** como interface de comunicação entre o cliente (ou usuário) e a camada de transação *blockchain* pode ser motivada por sua simplicidade, padronização e ampla compatibilidade com múltiplas plataformas. Um diagrama com os itens da arquitetura pode ser encontrada na Figura 8.

Figura 8 – Sistemas que compõem a arquitetura utilizada



Fonte: Elaborada pelo autor

Neste modelo:

- O cliente envia requisições HTTP à *API REST*, utilizando métodos como *POST* para iniciar transações.
- A *API* interpreta os dados, recupera ou publica no *Ganache* (*Ethereum* local). Publicar adiciona informações novas a *blockchain*, como contas ou transações.
- As respostas (incluindo *hashes* de transação, dados de retorno e eventuais erros) são devolvidas ao cliente de forma estruturada.
- Também é possível criar contas para determinar usuários diferentes com um método *POST* com essa responsabilidade.

Foi decidido executar a *blockchain Ethereum Ganache* em um container do tipo docker, mantendo (como foi discutido na sessão sobre docker e containers) uma arquitetura bem comum para microsserviços e facilitando a implantação.

É necessário salientar que o sistema não possui um banco de dados propriamente dito. Todas as informações são mantidas na própria *blockchain*, o que garante todas as vantagens previamente mencionadas sobre a utilização da *blockchain* quanto a segurança e privacidade, tão importantes em transações econômicas.

3.4 APLICAÇÃO DETALHADA

Foi desenvolvida uma *API RESTful* feita com *FastAPI* para interagir com a *blockchain Ethereum* (*Ganache*) para aplicações de *Smart Grid*. Esta API permite gerenciar contas *Ethereum* e realizar transações em um ambiente de *blockchain* local ou simulado.

3.4.1 Visão Geral e arquitetura

Esta API foi desenvolvida para fornecer uma interface simples e eficiente para interagir com uma *blockchain Ethereum* local. A arquitetura do sistema permite operações como criação de contas, transferências de *ETH* e consulta de histórico de transações.

A arquitetura proposta para a solução desenvolvida consiste em um modelo modular de quatro camadas, estruturado para promover escalabilidade, manutenção facilitada e clara separação de responsabilidades. Essa estrutura visa conectar clientes de forma segura e eficiente à *blockchain Ethereum*, operando localmente via *Ganache* em ambiente de simulação. A seguir, são descritas detalhadamente as camadas da arquitetura:

- **Camada 1: Cliente HTTP (*Frontend*)**
  Representa a aplicação cliente responsável por iniciar chamadas HTTP para a API. Pode ser composta por uma interface web, aplicativo móvel ou sistema automatizado. As requisições são estruturadas em formato *JSON*, utilizando métodos como *GET*, *POST* e *PUT* para acessar funcionalidades como criação de contas, consulta de saldo e execução de transações.

- **Camada 2: *FastAPI* API (*Backend*)**
  Desenvolvida com o *framework FastAPI*, esta camada expõe *endpoints RESTful* e atua como intermediária entre o cliente e o serviço de *blockchain*. Além de processar e validar as requisições recebidas, a API fornece documentação interativa utilizando o *Swagger* UI e está preparada para integrar-se com diversos sistemas externos por meio de uma interface padronizada.

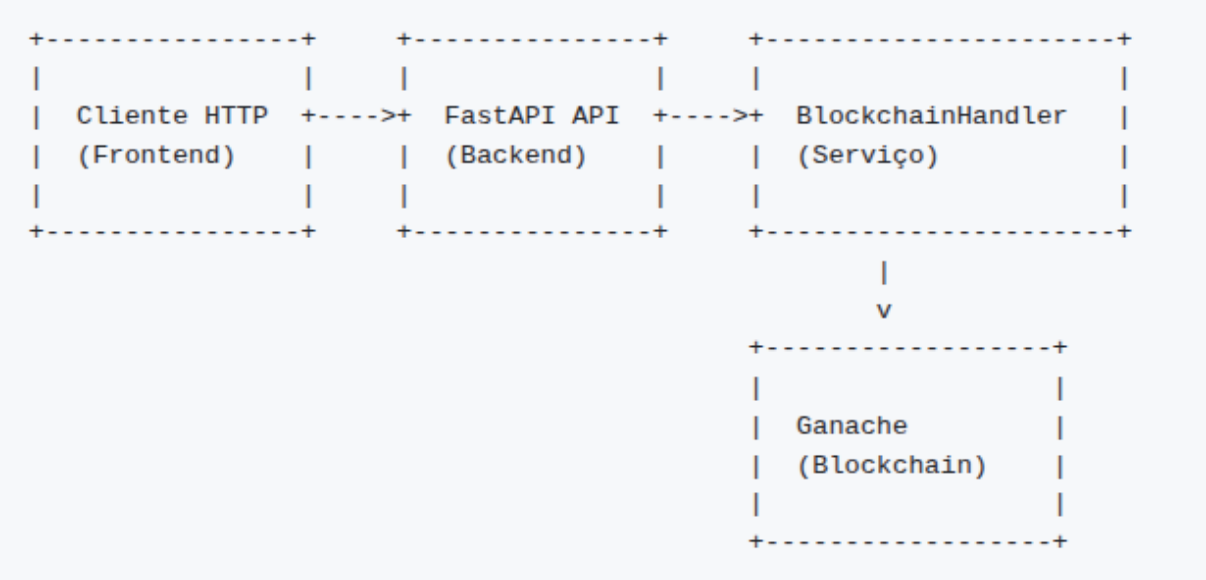
- **Camada 3: *BlockchainHandler* (Serviço de Aplicação)**
  Este módulo concentra a lógica de negócio que gerencia a interação com a *blockchain*. Utilizando a biblioteca *Web3.py*, o *BlockchainHandler* abstrai a complexidade técnica de operações como criação de transações, geração de carteiras, leitura de blocos e consultas de estado. Essa separação permite encapsular funcionalidades específicas da *blockchain* e expô-las de forma organizada à API.

- **Camada 4: *Ganache* (Camada *Blockchain*)**
  A instância local da *blockchain Ethereum* é executada através do *Ganache*, uma ferramenta voltada para desenvolvimento e testes. *Ganache* permite controlar o

ambiente da cadeia, manipulando blocos, transações e contas simuladas. Essa camada registra todas as operações de forma imutável e cronológica, funcionando como sistema de persistência descentralizado durante a prototipagem.

Essa arquitetura (Figura 9) demonstra flexibilidade na integração de novas funcionalidades, como contratos inteligentes, suporte a múltiplos *tokens*, e orquestração de microserviços. Além disso, oferece uma base sólida para migração futura para redes *blockchain* reais, como *Goerli*, *Sepolia* ou redes permissionadas. A separação entre interface, lógica de negócio e persistência em *blockchain* garante robustez e facilita a manutenção do sistema desenvolvido.

Figura 9 – Arquitetura geral implementada



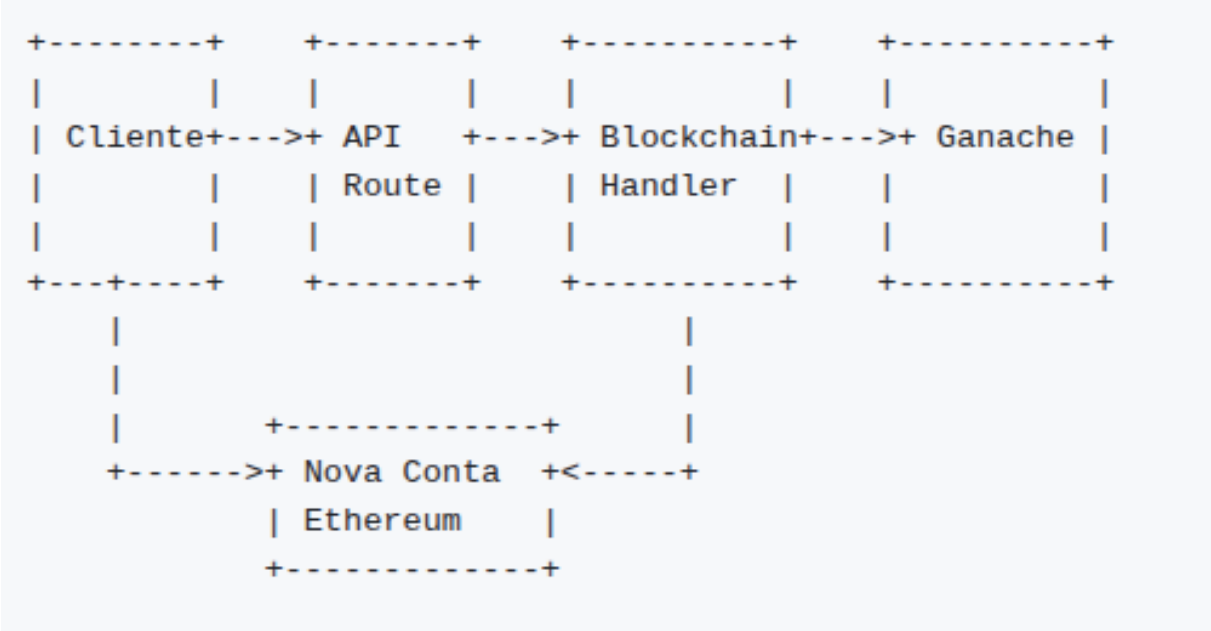
Fonte: Elaborada pelo autor

3.4.2 Funcionalidades

- Gerenciamento de Contas: Criar novas contas *Ethereum*, listar contas existentes, verificar saldos
- Transações: Enviar ETH entre contas *Ethereum* através da *blockchain*
- Histórico: Consultar histórico completo de transações de uma conta
- API *RESTful*: Interface padronizada seguindo princípios *REST*
- Documentação Interativa: *Swagger* UI para testar todos os *endpoints* de forma visual

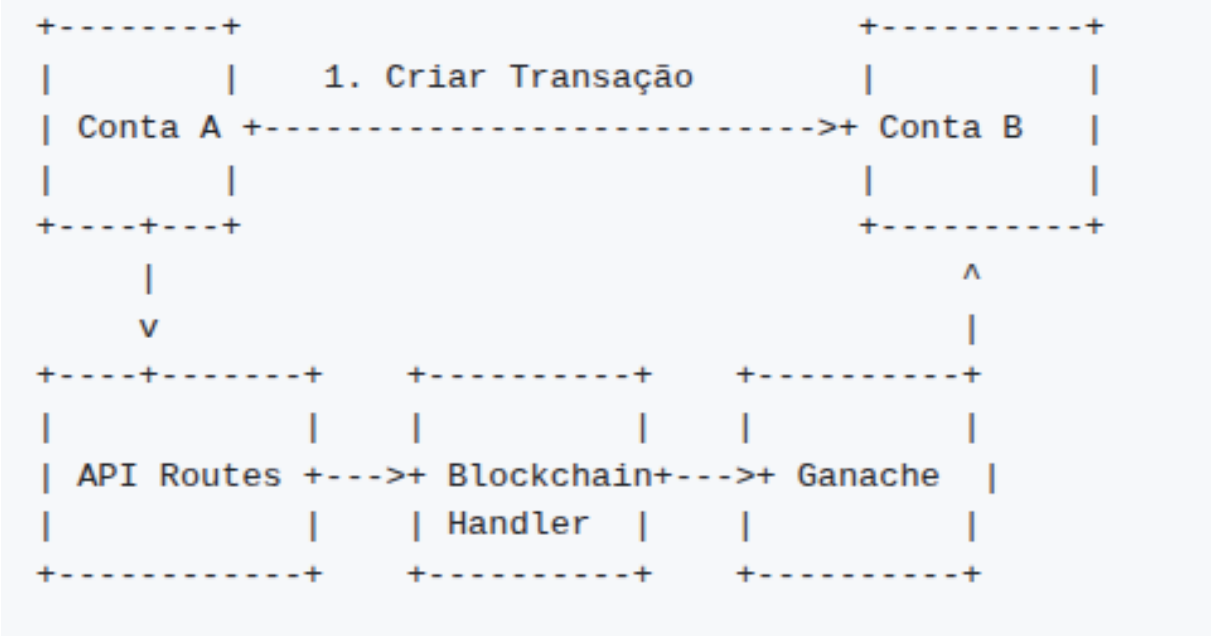
O fluxo para criação de conta pode ser vista no diagrama da figura 10, assim como o de transação pode ser visto na figura 11.

Figura 10 – Criação de conta



Fonte: Elaborada pelo autor

Figura 11 – Transação



Fonte: Elaborada pelo autor

3.4.3 *Endpoints* Detalhados da API *RESTful*

A seguir, são descritos os principais *endpoints* implementados na API *RESTful* para integração com a blockchain *Ethereum* local via *Ganache*. A estrutura segue o padrão de versionamento */api/v1* e está organizada em dois grupos principais: **Contas** e **Transações**.

3.4.3.1 Contas

- *GET /api/v1/accounts*
  Lista todas as contas disponíveis na rede local.

  Resposta:

```
[
  {
    "address": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
    "balance": 100.0
  }
]
```

- *GET /api/v1/accounts/{address}*
  Obtém detalhes de uma conta específica, incluindo histórico de transações.

  Parâmetros: *address* — Endereço da conta *Ethereum*.

- *POST /api/v1/accounts*
  Cria uma nova conta *Ethereum*.

  Corpo (opcional):

```
{
  "password": "senha_opcional"
}
```

  Resposta:

```
{
  "address": "0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199",
  "private_key": "0xdf57089febbacf7ba0bc227dafbffa9fc08a93fdc68e1e42411a14
}
```

- *POST /api/v1/accounts/{address}/fund*
  Adiciona ETH à conta informada.

  Parâmetros: *address* — Endereço da conta a ser financiada.

  Corpo:

```
{
  "amount": 10.0
}
```

Resposta:

```
{
  "address": "0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199",
  "previous_balance": 0.0,
  "new_balance": 10.0,
  "amount_added": 10.0,
  "transaction_hash": "0x5d4ea3a5052b30b29d62f5b82d97cf4b537eb5fcc652714b9
  "status": "success"
}
```

3.4.3.2 Transações

- *POST /api/v1/transactions*
  Envia uma transação de uma conta para outra.

  Corpo:

```
{
  "from_address": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
  "to_address": "0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199",
  "amount": 1.5,
  "private_key": "0xdf57089febbacf7ba0bc227dafbffa9fc08a93fdc68e1e42411a14
}
```

  Resposta: Detalhes da transação executada com hash, status e confirmação.

- *GET /api/v1/transactions/{address}*
  Obtém o histórico de transações de uma conta (transações enviadas e recebidas).

  Parâmetros: *address* — Endereço da conta *Ethereum*.

  Resposta: Lista de transações vinculadas ao endereço consultado.

4 CONCLUSÕES

Este trabalho teve como principal objetivo investigar soluções tecnológicas inovadoras para a modernização do setor energético brasileiro, com foco na integração entre **smart grids** e **blockchain**. A abordagem metodológica adotada envolveu etapas aprofundadas de levantamento bibliográfico, análise crítica do sistema elétrico nacional, estudo técnico das redes inteligentes, modelagem de arquiteturas blockchain e definição de estratégias para negociação descentralizada de energia.

Inicialmente, foi realizado um mapeamento conceitual das *smart grids*, considerando sua infraestrutura técnica, benefícios operacionais e barreiras de adoção. Em seguida, foram estudadas diferentes abordagens de aplicação da tecnologia *blockchain* no setor energético.

Pode ser compreendido os desafios e vantagens da implantação de um sistema moderno e tecnológico como *Smart Grids* em um ambiente real (FALCÃO, 2010). *Smart Grid* é um conceito que une várias tecnologias e é dependente do desenvolvimento de várias ferramentas necessárias para tornar a compra, venda e transmissão de energia no mercado livre segura (CCEE, 2025; FALCÃO, 2010).

Percebeu-se que não há uma maneira de desenvolver um ambiente de *Smart Grid*, mas sim várias opções possíveis (IJAYAPRIYA TAMILMARAN; KOTHARI, 2011; FALCÃO, 2010). Ficou evidente que existe discussão sobre *blockchains* integrarem esse tipo de distribuição de energia (ALLADI et al., 2019).

Como etapa de validação prática, foi desenvolvida uma API *RESTful*, integrada a uma instância *Ethereum* local executada via *Ganache* rodando em um container *Docker*. Esta API permite criar contas, consultar saldos e executar transações energéticas simuladas, adotando uma arquitetura de três camadas: interface, serviço e armazenamento em *blockchain*.

Sobre o trabalho realizado e a ferramenta desenvolvida, é interessante mencionar que:

4.1 DESEMPENHO E ESCALABILIDADE

A arquitetura descentralizada baseada em *Ethereum* impõe limitações de desempenho que afetam a escalabilidade da solução:

- **Latência nas Confirmações**: O tempo de confirmação de uma transação pode variar entre segundos e minutos, dependendo da congestão da rede e do valor do *gas* oferecido, o que pode comprometer respostas em tempo real em cenários críticos.

- **Requisitos de Armazenamento**: O custo de armazenamento na *blockchain* é elevado. Foram adotadas estratégias para minimizar a persistência de dados na cadeia, restringindo os registros apenas ao necessário para manter integridade e auditabilidade do sistema.

4.2 VANTAGENS DA IMPLEMENTAÇÃO

4.2.1 Transparência e Auditabilidade

A adoção da tecnologia *blockchain* na proposta apresentada trouxe avanços significativos em termos de confiabilidade e rastreabilidade das operações:

- **Registro Imutável**: Todas as transações realizadas são armazenadas de forma cronológica e imutável na *blockchain Ethereum*, garantindo integridade histórica e impedindo modificações indevidas no registro de operações da rede elétrica inteligente.
- **Auditoria Facilitada**: A natureza transparente da *blockchain* permite que agentes reguladores, desenvolvedores e consumidores auditem livremente as transações realizadas, dispensando intermediários e sistemas centralizados para verificação de integridade.
- **Eliminação de Intermediários**: O modelo implementado viabiliza transações *peer-to-peer* sem a necessidade de entidades confiáveis intermediárias.

4.2.2 Segurança e Integridade de Dados

A infraestrutura *blockchain* oferece mecanismos robustos que fortalecem a proteção dos dados e garantem a autenticidade das operações:

- **Criptografia de Ponta**: A comunicação e autenticação são realizadas através de criptografia assimétrica da própria *blockchain*, utilizando chaves públicas e privadas, assegurando que somente os usuários autorizados possam realizar ações específicas na rede.
- **Validação por Consenso**: O mecanismo de consenso adotado garante que todas as transações sejam confirmadas por múltiplos nós validadores, aumentando a confiabilidade do sistema e reduzindo pontos únicos de falha.

4.2.3 Potencial para Mercados de Energia Descentralizados

A implementação realizada estabelece bases sólidas para o desenvolvimento de ecossistemas energéticos modernos:

- **Transações *Peer-to-Peer***: Usuários podem realizar negociações diretas de energia, sem intermediários, permitindo a construção de mercados locais e redes comunitárias com autonomia operacional.
- **Incentivos à Energia Renovável**: O registro rastreável das operações permite a criação de mecanismos de incentivo à produção e consumo de fontes renováveis, como certificados de energia limpa ou créditos de carbono.

4.3 ANÁLISE TÉCNICA DA IMPLEMENTAÇÃO

4.3.1 Avaliação da Arquitetura

A estrutura adotada segue princípios sólidos de engenharia de software:

- **Separação de Responsabilidades**: A divisão clara entre a camada de API, o serviço de manipulação da *blockchain* (*Blockchain Handler*) e a camada de armazenamento facilita manutenções e garante modularidade.
- **Abstração da Complexidade**: O programa possui um módulo para interação com a *blockchain* que encapsula as operações complexas, expondo apenas funcionalidades necessárias à interface, promovendo simplicidade sem comprometer segurança.

4.4 QUANTO A APLICABILIDADE DA IMPLEMENTAÇÃO COMO PROVA DE CONCEITO

O estudo desenvolvido ao longo deste trabalho, aliado à implementação técnica da API RESTful, configura uma prova de conceito funcional que valida a viabilidade de desenvolver, de forma acessível e modular, uma aplicação voltada para transações seguras para compra e venda de energia de forma descentralizada no contexto de *smart grids*.

A abordagem adotada, baseada na arquitetura *RESTful* e no uso de tecnologias amplamente difundidas como containers *Docker* demonstrou que é possível construir uma solução prática que permite a realização de compra e venda de energia elétrica entre pares. Por meio de operações expostas em *endpoints* HTTP padronizados, a API se comunica com uma instância local da *blockchain Ethereum*, instanciada via *Ganache*, registrando as transações de forma imutável e transparente.

Em síntese, este trabalho demonstrou que é possível, com recursos acessíveis e arquitetura bem definida, construir soluções tecnológicas que viabilizam transações inteligentes e descentralizadas que podem ser aplicadas na compra de energia elétrica (ALLADI et al., 2019).

4.5 RECOMENDAÇÕES PARA EVOLUÇÃO FUTURA

Como forma de evoluir a ferramenta desenvolvida, destacam-se:

- **Contratos Inteligentes Específicos**: O desenvolvimento de *smart contracts* especializados para funções energéticas, como medição automatizada, faturamento dinâmico e compensação de geração excedente, permitirá maior automação e adequação do sistema à lógica operacional de uma *smart grid* real.
- **Aplicação Web**: Para facilitar a interface com o usuário, pode-se desenvolver uma aplicação WEB, como um site, que faz as requisições HTTP para os *endpoints* criados neste projeto. Isso facilitaria muito o uso da solução para usuários comuns.

4.6 CONSIDERAÇÕES FINAIS

Os desafios técnicos enfrentados durante a implementação refletem a complexidade inerente à integração de tecnologias emergentes como *blockchain* com infraestruturas críticas como redes elétricas. A complexidade da integração com *blockchain*, limitações de desempenho e escalabilidade, e considerações de segurança representam obstáculos significativos que foram abordados de forma estratégica na solução analisada.

As vantagens proporcionadas pela implementação são igualmente significativas. A transparência e auditabilidade inerentes à tecnologia *blockchain* (FALCÃO, 2010), os elevados níveis de segurança e integridade de dados, a flexibilidade arquitetural e o potencial para habilitar mercados de energia descentralizados representam avanços em relação aos sistemas energéticos tradicionais.

A evolução futura da solução, conforme as recomendações apresentadas neste trabalho, tem o potencial de ampliar ainda mais estas vantagens, abordando as limitações atuais e explorando novas possibilidades tecnológicas e de mercado.

Em última análise, a implementação analisada estabelece uma base para futuras implementações construírem uma ferramenta que prática e utilizável pelo público geral em transações modernas no contexto de compra e venda de energia de maneira segura utilizando *blockchain* no contexto de *Smart Grids*.

REFERÊNCIAS

AHRAM, Tareq et al. Blockchain Technology Innovations. In: 2017 IEEE Technology & Engineering Management Conference (TEMSCON). IEEE, 2017. P. 137–141. DOI: 10.1109/TEMSCON.2017.7998367. Disponível em: <https://ieeexplore.ieee.org/document/7998367.%20Acessado%20em:%2024%20jun.%202025>.

ALLADI, Tejasvi et al. Blockchain in Smart Grids: A Review on Different Use Cases. **Sensors**, MDPI, v. 19, n. 22, p. 4862, 2019. DOI: 10.3390/s19224862.

AMAZON. 2024. Disponível em: <aws.amazon.com/pt/what-is/api/%20acessado%20em%2001%20de%20Julho%20de%202025>.

ANDERSON, Charles. Docker [Software engineering]. **IEEE Software**, IEEE, v. 32, n. 3, p. 102–c3, mai. 2015. DOI: 10.1109/MS.2015.62.

BARI A JIANG J, et al. Challenges in the Smart Grid Applications: An Overview. **International Journal of Distributed Sensor Networks; 10(2)**, 2014.

CCEE. **Estudo CCEE mostra crescimento de 3,9 no consumo de energia elétrica no Brasil em 2024**. 2024. Disponível em: <https://www.ccee.org.br/pt/web/guest/-/estudo-da-ccee-mostra-crescimento-de-3-9-no-consumo-de-energia-eletrica-no-brasil-em-2024>.

_____. **Mercado Livre**. 2025. Disponível em: <https://www.ccee.org.br/mercado-livre-acl>.

DOCKER. 2024. Disponível em: <https://www.docker.com/resources/what-container/,%20acessado%20em%2001%20de%20Julho%20de%202025>.

ETHEREUM. 2025. Disponível em: <https://ethereum.org/en/developers/>.

FALCÃO, Djalma M. **Integração de Tecnologias para Viabilização da Smart Grid**. 2010.

GANACHE. 2025. Disponível em: <https://archive.trufflesuite.com/ganache/>.

HAMIDI, V.; SMITH, K. S.; WILSON, R. C. Smart Grid technology review within the Transmission and Distribution sector. In: IEEE. 2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe). Gothenburg, Sweden: IEEE, out. 2010. P. 1–8. DOI: 10.1109/ISGTEUROPE.2010.5638950.

HELLWIG, Daniel; KARLIC, Goran; HUCHZERMEIER, Arnd. **Build Your Own Blockchain: A Practical Guide to Distributed Ledger Technology**. 1. ed.: Springer, 2020. (Management for Professionals). DOI: 10.1007/978-3-030-40690-5.

HYPERLEDGER. 2025. Disponível em: <https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html>.

IJAYAPRIYA TAMILMARAN; KOTHARI, Dwarkadas Pralhadas. Smart Grid: An Overview. **Smart Grid and Renewable Energy, v. 2, n. 4, p. 305–311**, 2011. Disponível em: <https://www.scirp.org/journal/paperinformation?paperid=8269.%20Acesso%20em:%2024%20jun.%202025>.

MASSÉ, Mark. **REST API Design Rulebook**. Sebastopol, CA: O'Reilly Media, Inc., 2012. Accessed: 2025-06-24. ISBN 9781449310509.

MOLOLOTH, Vidya Krishnan; SAGUNA, Saguna; ÅHLUND, Christer. Blockchain and Machine Learning for Future Smart Grids: A Review. **Energies**, MDPI, v. 16, n. 1, p. 528, 2023. Accessed: 2025-06-24. DOI: 10.3390/en16010528. Disponível em: <https://doi.org/10.3390/en16010528>.

NAKAMOTO, Satoshi. **Bitcoin: A Peer-to-Peer Electronci Cash System**. Disponível em: <https://bitcoin.org/bitcoin.pdf.%20Acessado%20em:%2024%20jun.%202025>.

PAUL, Shuva et al. Approaching to the Smartest technological use the whole world is being aware of using the smartest technology of generating and distributing power throughout the country in a simple but effective system. In: IEEE. 2014 1st International Conference on Non Conventional Energy (ICONCE). IEEE, 2014. P. 1–5. DOI: 10.1109/ICONCE.2014.6808719.

PIERRO, Massimo Di. What Is the Blockchain? **Computing in Science & Engineering**, IEEE, v. 19, n. 5, p. 92–95, 2017. DOI: 10.1109/MCSE.2017.3421554. Disponível em: <https://ieeexplore.ieee.org/document/8024092.%20Acessado%20em:%2024%20jun.%202025>.

SUNNY, Farhana Akter et al. **A Systematic Literature Review on Blockchain Technology for Smart Applications**. v. 10. IEEE, 2022. P. 59155–59177. Accessed: 2025-06-24. DOI: 10.1109/ACCESS.2022.3179690. Disponível em: <https://ieeexplore.ieee.org/document/9786930>.

APÊNDICE A – CÓDIGO DA FERRAMENTA DESENVOLVIDA:

Código A.1 – Ponto de entrada no código

```python
from fastapi import FastAPI, HTTPException, Depends
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, Field
from typing import List, Dict, Any, Optional
from datetime import datetime
import uvicorn
from services.blockchain_handler import BlockchainHandler

# Iniciar a API FastAPI
app = FastAPI(
    title="Smart Grid API",
    description="API para interação com blockchain Ethereum para aplicações de Smart Grid",
    version="1.0.0"
)

# Configurar CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Inicializar o handler do blockchain
blockchain = BlockchainHandler()

# Modelos Pydantic para validação de dados
class TransactionCreate(BaseModel):
    from_address: str = Field(..., description="Endereço da conta de origem")
    to_address: str = Field(..., description="Endereço da conta de destino")
    amount: float = Field(..., description="Quantidade de ETH a ser transferida")
    private_key: str = Field(..., description="Chave privada da conta de origem")

class AccountCreate(BaseModel):
    password: Optional[str] = Field(None, description="Senha opcional para proteger a conta")

@app.get("/")
async def root():
```

```
    return {"message": "Smart Grid API funcionando!"}

# Incluir os endpoints de contas e transações
from routes.v1 import accounts, transactions

app.include_router(
    accounts.router,
    prefix="/api/v1/accounts",
    tags=["accounts"]
)

app.include_router(
    transactions.router,
    prefix="/api/v1/transactions",
    tags=["transactions"]
)

if __name__ == "__main__":
    uvicorn.run("app:app", host="0.0.0.0", port=5000, reload=True)
```

Código A.2 – Camada de domínio do código que lida com a blockchain

```
from web3 import Web3
from typing import List, Dict, Any, Optional
from eth_account import Account
import os
from dotenv import load_dotenv
import json
import random
import time
from datetime import datetime, timedelta

load_dotenv()

class BlockchainHandler:
    """
    Gerenciador de interações com blockchain Ethereum.

    Fornece uma interface para interagir com blockchain Ethereum através do Ganache.
    Suporta dois modos de operação:
    1. Modo de conexão real: Conecta-se ao Ganache e interage diretamente com o blockchain
    2. Modo de simulação: Simula operações blockchain quando não é possível conectar ao Ganache
    """

    def __init__(self):
```

```
24          """
25          Inicializa o handler do blockchain.
26
27          Tenta conectar-se ao Ganache na URL http://localhost:8545.
28          Se a conexão falhar, ativa automaticamente o modo de simulação.
29          No modo de simulação, cria contas e transações simuladas em memória.
30          """
31          self.simulation_mode = False
32          try:
33              self.w3 = Web3(Web3.HTTPProvider('http://localhost:8545'))
34              if not self.w3.is_connected():
35                  print("Aviso: Falha ao conectar com o blockchain. Executando em
    modo de simulação.")
36                  self.simulation_mode = True
37              else:
38                  print(f"Conectado ao blockchain: {self.w3.is_connected()}")
39                  print(f"Chain ID: {self.w3.eth.chain_id}")
40          except Exception as e:
41              print(f"Erro ao conectar com o blockchain: {e}")
42              print("Executando em modo de simulação.")
43              self.simulation_mode = True
44
45          if self.simulation_mode:
46              self.simulated_accounts = [
47                  {
48                      'address': f"0x{i}{'0' * 39}",
49                      'balance': 100.0,
50                      'private_key': f"0x{i}{'1' * 63}"
51                  }
52                  for i in range(1, 11)
53              ]
54              self.simulated_transactions = []
55
56      def list_all_accounts(self) -> List[Dict[str, Any]]:
57          """
58          Lista todas as contas disponíveis no blockchain.
59
60          Retorna uma lista de todas as contas com seus respectivos endereços e
    saldos.
61          No modo real, obtém as contas diretamente do Ganache.
62          No modo de simulação, retorna as contas simuladas em memória.
63
64          Returns:
65              List[Dict[str, Any]]: Lista de dicionários contendo endereço e saldo de
     cada conta
66          """
67          if self.simulation_mode:
```

```
68              return self.simulated_accounts
69
70          accounts = []
71          for address in self.w3.eth.accounts:
72              balance = self.w3.eth.get_balance(address)
73              accounts.append({
74                  'address': address,
75                  'balance': self.w3.from_wei(balance, 'ether'),
76              })
77          return accounts
78
79      def get_account(self, address: str) -> Optional[Dict[str, Any]]:
80          """
81          Obtém informações de uma conta específica pelo seu endereço.
82
83          Busca os detalhes de uma conta blockchain a partir do seu endereço.
84          No modo real, consulta o saldo diretamente do Ganache.
85          No modo de simulação, busca nas contas simuladas em memória.
86
87          Args:
88              address (str): Endereço da conta Ethereum a ser consultada
89
90          Returns:
91              Optional[Dict[str, Any]]: Dicionário com informações da conta (endereço
     e saldo)
92                                        ou None se a conta não for encontrada
93          """
94          if self.simulation_mode:
95              for account in self.simulated_accounts:
96                  if account['address'].lower() == address.lower():
97                      return account
98              return None
99
100         if not self.w3.is_address(address):
101             return None
102
103         try:
104             balance = self.w3.eth.get_balance(address)
105             return {
106                 'address': address,
107                 'balance': self.w3.from_wei(balance, 'ether'),
108             }
109         except Exception as e:
110             print(f"Erro ao obter informações da conta {address}: {e}")
111             return None
112
113     def create_account(self, password: Optional[str] = None) -> Dict[str, Any]:
```

```
114         """
115         Cria uma nova conta Ethereum.
116 
117         Gera um novo par de chaves Ethereum e retorna o endereço e a chave privada.
118         No modo real, utiliza a biblioteca eth_account para criar uma conta real.
119         No modo de simulação, gera um novo endereu00e7o e chave privada simulados.
120 
121         Args:
122             password (Optional[str], optional): Senha para proteger a chave privada
    .
123                                                 Não implementado completamente nesta
    versão.
124                                                 Defaults to None.
125 
126         Returns:
127             Dict[str, Any]: Dicionário contendo o endereço e a chave privada da
    nova conta
128 
129         Note:
130             Em ambiente de produção, a chave privada nunca deve ser retornada
    diretamente.
131             Idealmente, deve ser criptografada ou armazenada de forma segura.
132         """
133         if self.simulation_mode:
134             new_id = len(self.simulated_accounts) + 1
135             new_account = {
136                 'address': f"0x{new_id}{'0' * 39}",
137                 'balance': 100.0,
138                 'private_key': f"0x{new_id}{'1' * 63}"
139             }
140             self.simulated_accounts.append(new_account)
141             return {
142                 'address': new_account['address'],
143                 'private_key': new_account['private_key'],
144             }
145 
146         account = Account.create()
147         private_key = account.key.hex()
148         address = account.address
149 
150         return {
151             'address': address,
152             'private_key': private_key,
153         }
154 
155     def send_transaction(self, from_address: str, to_address: str,
156                          amount: float, private_key: str) -> Dict[str, Any]:
```

```
157         """
158         Envia uma transação de ETH de uma conta para outra.
159 
160         Transfere uma quantidade específica de Ether entre duas contas Ethereum.
161         Requer a chave privada da conta de origem para assinar a transação.
162         Realiza validações de endereço, saldo e chave privada antes da execução.
163 
164         Args:
165             from_address (str): Endereço da conta de origem
166             to_address (str): Endereço da conta de destino
167             amount (float): Quantidade de Ether a ser transferida
168             private_key (str): Chave privada da conta de origem para assinar a
    transação
169 
170         Returns:
171             Dict[str, Any]: Dicionário contendo detalhes da transação realizada
172 
173         Raises:
174             ValueError: Se os endereços forem inválidos, saldo insuficiente, ou
    chave privada incorreta
175         """
176         if self.simulation_mode:
177             from_account = None
178             to_account = None
179 
180             for account in self.simulated_accounts:
181                 if account['address'].lower() == from_address.lower():
182                     from_account = account
183                 if account['address'].lower() == to_address.lower():
184                     to_account = account
185 
186             if not from_account or not to_account:
187                 raise ValueError("Endereço inválido")
188 
189             if from_account['balance'] < amount:
190                 raise ValueError(f"Saldo insuficiente. Disponível: {from_account['
    balance']} ETH")
191 
192             if from_account['private_key'] != private_key:
193                 raise ValueError("Chave privada inválida")
194 
195             from_account['balance'] -= amount
196             to_account['balance'] += amount
197 
198             tx_hash = f"0x{random.randint(0, 10**50):x}"
199             transaction = {
200                 'transaction_hash': tx_hash,
```

```
201                 'from': from_address,
202                 'to': to_address,
203                 'amount': amount,
204                 'block_number': len(self.simulated_transactions) + 1,
205                 'status': 'success',
206                 'timestamp': int(time.time())
207             }
208 
209             self.simulated_transactions.append(transaction)
210 
211             return {
212                 'transaction_hash': tx_hash,
213                 'from_address': from_address,
214                 'to_address': to_address,
215                 'amount': amount,
216                 'block_number': len(self.simulated_transactions) + 1,
217                 'status': 'success'
218             }
219 
220         if not self.w3.is_address(from_address) or not self.w3.is_address(
    to_address):
221             raise ValueError("Endereço inválido")
222 
223         amount_wei = self.w3.to_wei(amount, 'ether')
224 
225         # Verificar saldo
226         balance = self.w3.eth.get_balance(from_address)
227         if balance < amount_wei:
228             raise ValueError(f"Saldo insuficiente. Disponível: {self.w3.from_wei(
    balance, 'ether')} ETH")
229 
230         # Obter nonce (número sequencial de transações da conta)
231         nonce = self.w3.eth.get_transaction_count(from_address)
232 
233         try:
234             # Definir parâmetros de gas mais seguros para evitar o erro 'base fee
    exceeds gas limit'
235             # Usar um limite de gas fixo maior para transferências simples
236             gas_limit = 100000  # Usar um valor maior que o padrão de 21000
237 
238             # Usar um preço de gas fixo e conservador
239             # A definição explícita evita problemas com fees dinâmicos no Ganache
240             gas_price = 20000000000  # 20 Gwei, um valor seguro para testes no
    Ganache
241 
242             tx_params = {
243                 'from': from_address,
```

```
244                 'to': to_address,
245                 'value': amount_wei,
246                 'gas': gas_limit,
247                 'gasPrice': gas_price,
248                 'nonce': nonce,
249                 'chainId': self.w3.eth.chain_id
250             }
251 
252             # Assinar a transação com a chave privada
253             signed_tx = self.w3.eth.account.sign_transaction(tx_params, private_key
    )
254 
255             # Enviar a transação assinada
256             tx_hash = self.w3.eth.send_raw_transaction(signed_tx.rawTransaction)
257 
258             # Aguardar a confirmação da transação
259             tx_receipt = self.w3.eth.wait_for_transaction_receipt(tx_hash)
260 
261             # Retornar com nomes de campos compatíveis com TransactionResponse
262             return {
263                 'transaction_hash': tx_hash.hex(),
264                 'from_address': from_address,
265                 'to_address': to_address,
266                 'amount': amount,
267                 'block_number': tx_receipt['blockNumber'],
268                 'status': 'success' if tx_receipt['status'] == 1 else 'failed'
269             }
270         except Exception as e:
271             # Capturar e relançar qualquer erro durante o processo de transação
272             raise ValueError(f"Erro ao processar transação: {str(e)}")
273 
274 
275     def fund_account(self, address: str, amount: float) -> Dict[str, Any]:
276         """
277         Adiciona ETH a uma conta Ethereum.
278 
279         Em ambientes de teste, é comum precisar financiar contas reciém-criadas.
280         No modo real, usa a primeira conta do Ganache (que geralmente tem muito ETH
    ) para enviar fundos.
281         No modo de simulação, simplesmente aumenta o saldo da conta.
282 
283         Args:
284             address (str): Endereço da conta Ethereum a ser financiada
285             amount (float): Quantidade de ETH a ser adicionada
286 
287         Returns:
288             Dict[str, Any]: Detalhes da operação realizada no formato esperado pelo
```

```
    FundAccountResponse

        Raises:
            ValueError: Se o endereço for inválido ou ocorrer erro na transação
        """
        if self.simulation_mode:
            # Verificar se a conta existe
            account = None
            for acc in self.simulated_accounts:
                if acc['address'].lower() == address.lower():
                    account = acc
                    break

            if not account:
                raise ValueError("Conta não encontrada")

            # Registrar o saldo anterior
            previous_balance = account['balance']

            # Adicionar os fundos
            account['balance'] += amount

            # Gerar hash simulado para a transação
            tx_hash = f"0x{''.join(random.choices('0123456789abcdef', k=64))}"

            # Registrar a transação no histórico
            admin_account = self.simulated_accounts[0]  # Usamos a primeira conta
    como "admin" para o financiamento
            timestamp = datetime.now().timestamp()

            transaction = {
                'transaction_hash': tx_hash,
                'from': admin_account['address'],
                'to': address,
                'amount': amount,
                'block_number': len(self.simulated_transactions) + 1,
                'timestamp': timestamp,
                'status': 'success'
            }

            self.simulated_transactions.append(transaction)

            # Formatar resposta conforme esperado no FundAccountResponse
            return {
                'address': address,
                'previous_balance': previous_balance,
                'new_balance': account['balance'],
```

```
334                 'amount_added': amount,
335                 'transaction_hash': tx_hash,
336                 'status': 'success'
337             }
338 
339         # Modo real - usando Ganache
340         if not self.w3.is_address(address):
341             raise ValueError("Endereço inválido")
342 
343         try:
344             # No Ganache, vamos pegar a primeira conta (que geralmente tem muito
    ETH)
345             # e usar essa conta para financiar a nova conta
346             sender_address = self.w3.eth.accounts[0]
347             sender_private_key = "0
    x4f3edf983ac636a65a842ce7c78d9aa706d3b113bce9c46f30d7d21715b23b1d"  # Chave
    privada da primeira conta no Ganache deterministic mode
348 
349             # Verificar saldo do remetente
350             sender_balance = self.w3.eth.get_balance(sender_address)
351             amount_wei = self.w3.to_wei(amount, 'ether')
352 
353             if sender_balance < amount_wei:
354                 raise ValueError(f"Conta do financiador não tem ETH suficiente.
    Disponível: {self.w3.from_wei(sender_balance, 'ether')} ETH")
355 
356             # Obter o saldo atual do destinatário para comparar depois
357             prev_balance = self.w3.eth.get_balance(address)
358             prev_balance_eth = self.w3.from_wei(prev_balance, 'ether')
359 
360             # Preparar e enviar a transação
361             transaction = self.send_transaction(
362                 from_address=sender_address,
363                 to_address=address,
364                 amount=amount,
365                 private_key=sender_private_key
366             )
367 
368             # Obter o novo saldo
369             new_balance = self.w3.eth.get_balance(address)
370             new_balance_eth = self.w3.from_wei(new_balance, 'ether')
371 
372             # Formatar resposta conforme esperado no FundAccountResponse
373             return {
374                 'address': address,
375                 'previous_balance': prev_balance_eth,
376                 'new_balance': new_balance_eth,
```

```
377                 'amount_added': amount,
378                 'transaction_hash': transaction['transaction_hash'],
379                 'status': transaction['status']
380             }
381         except Exception as e:
382             raise ValueError(f"Erro ao financiar conta: {str(e)}")
383 
384     def get_transaction_history(self, address: str) -> List[Dict[str, Any]]:
385         """
386 {{ ... }}
387 
388         Busca todas as transações (enviadas e recebidas) associadas a um
    determinado endereço.
389         No modo real, consulta os blocos da blockchain para encontrar transações
    relacionadas.
390         No modo de simulação, pesquisa no registro de transações simuladas em memó
    ria.
391 
392         Args:
393             address (str): Endereço da conta Ethereum para consultar o histórico
394 
395         Returns:
396             List[Dict[str, Any]]: Lista de transações associadas ao endereço,
397                                   cada uma contendo detalhes como hash da transação
    ,
398                                   endereços de origem e destino, valor, etc.
399 
400         Raises:
401             ValueError: Se o endereço fornecido for inválido
402         """
403         if self.simulation_mode:
404             history = []
405             for tx in self.simulated_transactions:
406                 if tx['from'].lower() == address.lower() or tx['to'].lower() ==
    address.lower():
407                     history.append({
408                         'transaction_hash': tx['transaction_hash'],
409                         'from': tx['from'],
410                         'to': tx['to'],
411                         'amount': tx['amount'],
412                         'block_number': tx['block_number'],
413                         'timestamp': tx['timestamp'],
414                         'type': 'sent' if tx['from'].lower() == address.lower()
    else 'received'
415                     })
416             return history
417 
```

```
        if not self.w3.is_address(address):
            raise ValueError("Endereço inválido")

        latest_block = self.w3.eth.block_number
        transactions = []

        for block_num in range(max(0, latest_block - 1000), latest_block + 1):
            try:
                block = self.w3.eth.get_block(block_num, full_transactions=True)
                for tx in block.transactions:
                    if tx['from'] == address or tx['to'] == address:
                        transactions.append({
                            'transaction_hash': tx['hash'].hex(),
                            'from': tx['from'],
                            'to': tx['to'],
                            'amount': self.w3.from_wei(tx['value'], 'ether'),
                            'block_number': block_num,
                            'timestamp': block.timestamp,
                            'type': 'sent' if tx['from'] == address else 'received'
                        })
            except Exception as e:
                print(f"Erro ao processar bloco {block_num}: {e}")
                continue

        return transactions
```

Código A.3 – Rotas da API para criar e gerir contas

```
from fastapi import APIRouter, HTTPException, Depends, Path, Query, Body
from typing import List, Dict, Any, Optional
from pydantic import BaseModel, Field
from services.blockchain_handler import BlockchainHandler

router = APIRouter()
blockchain = BlockchainHandler()

class AccountCreate(BaseModel):
    """
    Modelo de dados para criação de uma nova conta.

    Permite a especificação opcional de uma senha para proteger a conta.
    """
    password: Optional[str] = Field(None, description="Senha opcional para proteger
     a conta")

class AccountResponse(BaseModel):
    """
    Modelo de dados para resposta com informações de uma conta.
```

```
20
21      Contém o endereço e saldo da conta Ethereum.
22      """
23      address: str
24      balance: float
25
26  class AccountCreateResponse(BaseModel):
27      """
28      Modelo de dados para resposta após a criação de uma conta.
29
30      Contém o endereço e a chave privada da nova conta criada.
31      """
32      address: str
33      private_key: str
34
35  @router.get("/", response_model=List[AccountResponse])
36  async def get_all_accounts():
37      """
38      Lista todas as contas disponíveis no blockchain.
39
40      Retorna uma lista com todas as contas registradas no blockchain,
41      incluindo seus endereços e saldos atuais.
42
43      Returns:
44          List[AccountResponse]: Lista de contas com seus respectivos endereços e
        saldos
45
46      Raises:
47          HTTPException: Se ocorrer algum erro durante a obtenção das contas
48      """
49      try:
50          accounts = blockchain.list_all_accounts()
51          return accounts
52      except Exception as e:
53          raise HTTPException(status_code=500, detail=f"Erro ao listar contas: {str(e
        )}")
54
55  @router.get("/{address}", response_model=Dict[str, Any])
56  async def get_account(address: str = Path(..., description="Endereço da conta
        Ethereum")):
57      """
58      Obtém informações detalhadas de uma conta específica e seu histórico de transaç
        ões.
59
60      Consulta os detalhes de uma conta Ethereum pelo seu endereço e retorna
61      tanto as informações da conta quanto seu histórico completo de transações.
62
```

```
63      Args:
64          address (str): Endereço da conta Ethereum a ser consultada
65
66      Returns:
67          Dict[str, Any]: Dicionário contendo informações da conta e histórico de
        transações
68
69      Raises:
70          HTTPException: Se a conta não for encontrada ou ocorrer um erro na consulta
71      """
72      try:
73          account = blockchain.get_account(address)
74          if not account:
75              raise HTTPException(status_code=404, detail="Conta não encontrada")
76
77          history = blockchain.get_transaction_history(address)
78
79          return {
80              "account": account,
81              "transactions": history
82          }
83      except HTTPException:
84          raise
85      except Exception as e:
86          raise HTTPException(status_code=500, detail=f"Erro ao obter informações da
        conta: {str(e)}")
87
88  @router.post("/", response_model=AccountCreateResponse)
89  async def create_account(account_data: Optional[AccountCreate] = None):
90      """
91      Cria uma nova conta no blockchain.
92
93      Gera um novo par de chaves Ethereum (endereço e chave privada) e retorna
94      as informações da nova conta criada.
95
96      Args:
97          account_data (Optional[AccountCreate]): Dados opcionais para criação da
        conta,
98                                                  como senha para proteger a chave
        privada
99
100     Returns:
101         AccountCreateResponse: Informações da nova conta criada (endereço e chave
        privada)
102
103     Raises:
104         HTTPException: Se ocorrer algum erro durante a criação da conta
```

```
105     """
106     try:
107         password = None
108         if account_data:
109             password = account_data.password
110 
111         new_account = blockchain.create_account(password=password)
112         return new_account
113     except Exception as e:
114         raise HTTPException(status_code=500, detail=f"Erro ao criar conta: {str(e)}
        ")
115 
116 # Modelo para a requisição de financiamento da conta
117 class FundAccountRequest(BaseModel):
118     """
119     Modelo de dados para financiar uma conta Ethereum.
120 
121     Contém a quantidade de ETH a ser adicionada à conta.
122     """
123     amount: float = Field(..., description="Quantidade de ETH a ser adicionada", gt
        =0)
124 
125 # Modelo para a resposta de financiamento da conta
126 class FundAccountResponse(BaseModel):
127     """
128     Modelo de dados para resposta após financiar uma conta.
129 
130     Contém os detalhes da operação e os novos saldos.
131     """
132     address: str
133     previous_balance: Optional[float]
134     new_balance: Optional[float]
135     amount_added: float
136     transaction_hash: str
137     status: str
138 
139 @router.post("/{address}/fund", response_model=FundAccountResponse)
140 async def fund_account(
141     address: str = Path(..., description="Endereço da conta Ethereum a ser
        financiada"),
142     fund_data: FundAccountRequest = Body(...)
143 ):
144     """
145     Adiciona ETH a uma conta Ethereum.
146 
147     Em ambientes de teste e desenvolvimento, é comum precisar financiar contas recé
        m-criadas.
```

```
148     Este endpoint permite adicionar uma quantidade específica de ETH a uma conta.
149
150     Args:
151         address (str): Endereço da conta Ethereum a ser financiada
152         fund_data (FundAccountRequest): Dados da requisição contendo a quantidade
    de ETH
153
154     Returns:
155         FundAccountResponse: Detalhes da operação realizada
156
157     Raises:
158         HTTPException: Se o endereço for inválido, a quantidade for inválida, ou
    ocorrer erro na transação
159     """
160     try:
161         # Verificar se a quantidade é válida
162         if fund_data.amount <= 0:
163             raise HTTPException(status_code=400, detail="A quantidade de ETH deve
    ser maior que zero")
164
165         # Chamar o método de financiamento no blockchain handler
166         result = blockchain.fund_account(address=address, amount=fund_data.amount)
167         return result
168     except ValueError as e:
169         raise HTTPException(status_code=400, detail=str(e))
170     except Exception as e:
171         raise HTTPException(status_code=500, detail=f"Erro ao financiar conta: {str
    (e)}")
```

Código A.4 – Rotas da API para criar e gerir transações

```
1  from fastapi import APIRouter, HTTPException, Path, Body
2  from typing import List, Dict, Any
3  from pydantic import BaseModel, Field
4  from services.blockchain_handler import BlockchainHandler
5
6  router = APIRouter()
7  blockchain = BlockchainHandler()
8
9  class TransactionCreate(BaseModel):
10     """
11     Modelo de dados para criação de uma nova transação.
12
13     Contém todos os campos necessários para executar uma transação entre contas
    Ethereum,
14     incluindo endereços de origem e destino, valor e chave privada para assinar a
    transação.
15     """
```

```
16      from_address: str = Field(..., description="Endereço da conta de origem")
17      to_address: str = Field(..., description="Endereço da conta de destino")
18      amount: float = Field(..., description="Quantidade de ETH a ser transferida")
19      private_key: str = Field(..., description="Chave privada da conta de origem")
20
21  class TransactionResponse(BaseModel):
22      """
23      Modelo de dados para resposta após a criação de uma transação.
24
25      Contém informações sobre a transação executada, incluindo o hash da transação,
26      endereços envolvidos, valor transferido, bloco e status.
27      """
28      transaction_hash: str
29      from_address: str
30      to_address: str
31      amount: float
32      block_number: int
33      status: str
34
35  @router.post("/", response_model=TransactionResponse)
36  async def send_transaction(transaction: TransactionCreate):
37      """
38      Envia uma transação entre duas contas Ethereum.
39
40      Recebe os dados da transação, valida os valores e envia a transação
41      para o blockchain através do BlockchainHandler.
42
43      Args:
44          transaction (TransactionCreate): Dados da transação a ser executada
45
46      Returns:
47          TransactionResponse: Detalhes da transação executada
48
49      Raises:
50          HTTPException: Se os endereços forem inválidos, o valor for inválido,
51                         ou ocorrer qualquer erro durante o processamento
52      """
53      try:
54          if not transaction.from_address or not transaction.to_address:
55              raise HTTPException(status_code=400, detail="Endereços de origem e
    destino são obrigatórios")
56
57          if transaction.amount <= 0:
58              raise HTTPException(status_code=400, detail="O valor da transação deve
    ser maior que zero")
59
60          result = blockchain.send_transaction(
```

```
            from_address=transaction.from_address,
            to_address=transaction.to_address,
            amount=transaction.amount,
            private_key=transaction.private_key
        )

        return result
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Erro ao enviar transação: {str(e)}")

@router.get("/{address}", response_model=List[Dict[str, Any]])
async def get_transaction_history(address: str = Path(..., description="Endereço da conta Ethereum")):
    """
    Obtém o histórico de transações de uma conta específica.

    Busca todas as transações (enviadas e recebidas) associadas ao endereço fornecido.

    Args:
        address (str): Endereço da conta Ethereum para consultar o histórico

    Returns:
        List[Dict[str, Any]]: Lista de transações associadas ao endereço

    Raises:
        HTTPException: Se o endereço for inválido ou ocorrer um erro na consulta
    """
    try:
        history = blockchain.get_transaction_history(address)
        return history
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Erro ao obter histórico de transações: {str(e)}")
```