CENTRO UNIVERSITÁRIO FEI

IGOR DOS SANTOS SILVA

# IMPLEMENTAÇÃO DE CONTRATOS INTELIGENTES PARA SEGUROS PARAMÉTRICOS DESCENTRALIZADOS

São Bernardo do Campo

2022

IGOR DOS SANTOS SILVA

# IMPLEMENTAÇÃO DE CONTRATOS INTELIGENTES PARA SEGUROS PARAMÉTRICOS DESCENTRALIZADOS

Trabalho de Conclusão de Curso apresentado ao Centro Universitário FEI para obtenção do título de Bacharel em Engenharia Elétrica com ênfase em Computadores. Orientado pelo Prof. Dr. Calebe de Paula Bianchini.

São Bernardo do Campo

2022

dos Santos Silva, Igor.
Implementação de contratos inteligentes para seguros paramétricos descentralizados / Igor dos Santos Silva. São Bernardo do Campo, 2022.
70 f. : il.

Trabalho de Conclusão de Curso - Centro Universitário FEI.
Orientador: Prof. Dr. Calebe de Paula Bianchini.

1. Seguros paramétricos. 2. Blockchain. 3. Ethereum. 4. Smart Contracts. 5. Web3. I. de Paula Bianchini, Calebe, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

IGOR DOS SANTOS SILVA

# IMPLEMENTAÇÃO DE CONTRATOS INTELIGENTES PARA SEGUROS PARAMÉTRICOS DESCENTRALIZADOS

Trabalho de Conclusão de Curso apresentado ao Centro Universitário FEI para obtenção do título de Bacharel em Engenharia Elétrica com ênfase em Computadores. Orientado pelo Prof. Dr. Calebe de Paula Bianchini.

Comissão julgadora

_____

Prof. Dr. Calebe de Paula Bianchini

Orientador

_____

Prof. Dr. Rodrigo Izidoro Tinini

Examinador

_____

Prof. Dr. Salvador Pinillos Gimenez

Examinador

São Bernardo do Campo

15/06/2022

Dedico este trabalho à minha família. Em especial, ao meu pai José Aparecido da Silva, o maior entusiasta da minha formação.

## AGRADECIMENTOS

Agradeço a Satoshi Nakamoto por apresentar ao mundo a tecnologia *Blockchain*, Vitalik Buterin pela construção da revolucionária rede *Ethereum* e a todos os desenvolvedores que contribuem com a comunidade *Open Source*. Agradeço ao meu orientador, Prof. Dr. Calebe de Paula Bianchini, pelas orientações ao longo desta jornada. Agradeço às pessoas que ajudaram diretamente com recomendações de ferramentas, testando a aplicação e com palavras de motivação. Dentre todas elas, vale destacar: Carlos Henrique, Cleyton Vinicius Esteves, Ian Pedro, Iago Davi, Isabela Stefany, José Aparecido (Zezinho), Lucas Gonçalves de Melo, Lucas Santos Dupim, Silvania Márcia (Marcinha) e Vinicius de Lima Costa (Macaé). É bem provável que eu deixe de citar alguém, mas fica aqui minha eterna gratidão a todos que colaboraram de alguma forma. Afinal, o trabalho apresentado aqui é resultado da colaboração direta ou indireta de muitas pessoas.

“A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.”

Albert Einstein

## RESUMO

A indústria de seguros é extremamente dependente de muitos processos que envolvem diferentes agentes. Adicione-se a isso a extrema centralização de poder das grandes companhias e o resultado é a promoção de produtos e a prestação de serviços não justa ao consumidor final. Em contrapartida, alguns modelos inovadores, como os seguros paramétricos, e a maturação de tecnologias emergentes estão causando revoluções e movimentando o setor. O presente trabalho de conclusão de curso discute algumas dessas tecnologias emergentes e propõe o uso delas para a criação de um novo modelo de negócio para a indústria de seguros: os seguros paramétricos descentralizados. Para atingir tal objetivo, por meio de uma prova de conceito que visa avaliar a viabilidade técnica, é implementada uma aplicação *web3* que permite a interação com *Smart Contracts* baseados em *Blockchain* na rede *Ethereum*. O contexto da aplicação é delimitado para a contratação de seguros paramétricos contra voos atrasados ou cancelados.



Palavras-chave: Seguros. Seguros Paramétricos. *Blockchain*. *Ethereum*. *Smart Contracts*. *Web3*. Prova de Conceito.

**ABSTRACT**

The insurance industry is extremely dependent on several processes involving different agents. When you combine this with the tremendous power concentration of large corporations, the outcome is a promotion of products and services that is unjust to the final consumer. In contrast, some innovation models, like parametric insurance, and the maturation of emerging technologies are making revolutions and moving the sector. The present paper discusses some of those emerging technologies and proposes their use to create a new business model to the insurance industry: the decentralized parametric insurances. To achieve this goal, through a proof of concept that wants to evaluate the technical viability, is implemented a web3 application that allows the interaction with smart contracts based on Blockchain on Ethereum network. The application context is delimited to contract parametric insurance for delayed or canceled flights.

Keywords: Insurances. Parametric Insurances. Blockchain. Ethereum. Smart Contracts. Web3. Proof of Concept.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## CÓDIGOS-FONTE

## LISTA DE ABREVIATURAS

| | |
|---|---|
| *API* | *Application Programming Interface* |
| *AWS* | *Amazon Web Services* |
| CPU | Unidade Central de Processamento |
| DAPP | Aplicação Descentralizada |
| *EVM* | *Ethereum Virtual Machine* |
| *HTTP* | *Hypertext Transfer Protocol* |
| *JSON* | *JavaScript Object Notation* |
| *PoS* | *Proof of Stack* |
| *PoW* | *Proof of Work* |
| *REST* | *Representational State Transfer* |
| RPC | Chamada de Procedimento Remoto |
| *SHA* | Algoritmo Hash Seguro |
| SUSEP | Superintendência de Seguros Privados |
| TPS | Transferência por Segundo |
| *TTS* | *Trusted Time-Stamping Service* |
| VESes | Sistemas de Execução Verificávies |

**SUMÁRIO**

# 1 INTRODUÇÃO

Nos últimos anos, com os avanços proporcionados pelo advento da *Internet*, o mundo se tornou cada vez mais globalizado. Hoje, devido às tecnologias de computação e comunicação, tem-se um grande intercâmbio cultural entre os países, o que restringe as barreiras físicas impostas entre eles e os tornam cada vez mais integrados.

Tendo-se o mundo cada vez mais integrado, torna-se possível, por consequência, oferecer uma vasta quantidade de serviços e produtos, além de possibilitar uma ampla concorrência do mercado, o que favorece os consumidores. Contudo, apesar do favorecimento aos consumidores no que tange à quantidade de produtos e serviços oferecidos, geralmente, não é entregue o maior valor possível a eles. Constantemente são veiculadas na mídia notícias que citam o uso de algoritmos para alterar o preço do produto de acordo com o perfil do internauta (INVESTE, 2022), algoritmos que mapeiam os hábitos e dão sugestões extremamente assertivas (POVO, 2022) etc. Além disso, a altíssima concentração de poder por parte das grandes companhias de tecnologia colocam a economia global em risco (TEK, 2022).

E se fosse possível reverter essa lógica? Se fosse possível construir produtos e serviços transparentes na qual há uma amplificação na entrega de valor ao consumidor final? A união de algumas manifestações do *Open Innovation* e aplicações descentralizadas parece fomentar um caminho viável para a construção de modelos de negócios em que o cliente é quem identifica o que tem mais valor para ele.

*Blockchain* pode ser resumido como um sistema distribuído e cronológico de transações descentralizado e de alta confiança (LOUKIL et al., 2021). *Smart contracts* baseados em *Blockchain*, por sua vez, são programas que rodam dentro de uma rede *Blockchain*. Este trabalho adota a rede *Ethereum*, uma manifestação do *Open Innovation*, que é uma rede *Blockchain* que permite a execução de *Smart Contracts* (ETHEREUM, 2022).

Fazendo-se uso das tecnologias citadas acima, tem-se espaço para muitas inovações no mercado. Alguns exemplos interessantes são: obter serviços de garantia financeira, pegar empréstimos de criptoativos, fazer investimentos para gerar renda passiva, realizar troca de ativos digitais de forma descentralizada etc (SANTANDER, 2022). O presente trabalho se concentra na indústria de seguros, visando demonstrar um modelo de negócio que é potencialmente explorável por qualquer pessoa, resultando na maximização de valor para todas as partes envolvidas.

## 1.1 MOTIVAÇÃO

A indústria de seguros no Brasil é multibilionária. Conforme as análises da Excelência em Gestão (2022a), em 2021 houve uma arrecadação de mais de 500 bilhões de reais. Ainda, de acordo com Excelência em Gestão (2022a), a tendência é de crescimento.

Apesar do tamanho do mercado, tem-se uma extrema concentração de poder. Segundo estudo divulgado pela Excelência em Gestão (2022b), a participação de mercado é concentrada em cinco empresas: BB Seguros (27%), Bradesco (20%), Caixa Seguros (19,2%), Zurich (8,9%) e Itaú (7,1%). Ou seja, estas 5 companhias citadas detêm mais de 80% do mercado.

A partir dos dados supramencionados, pode-se inferir que devido ao alto controle, abre-se possibilidade para construção de produtos e serviços que maximizam o valor entregue para a companhia e não para o consumidor final. Além disso, trata-se de uma indústria extremamente regulada e burocrática, o que dificulta a experiência do usuário final.

Tendo como enfoque a indústria de seguros paramétricos, a hipótese levantada por este trabalho é de que é possível construir um produto de seguro que use *Blockchain* e *Smart Contracts* baseados em *Blockchain*. Como resultado, espera-se a obtenção de um produto com governança descentralizada, cujas especificações serão acionadas de maneira autônoma com base nas regras definidas no *Smart Contract*. Se obtidos bons resultados a partir da hipótese levantada, configura-se o surgimento de um mercado de extremo potencial, com capacidade de entregar muito valor ao consumidor final.

Portanto, a grande motivação deste trabalho é mostrar caminhos alternativos que estão surgindo devido ao nascimento e adoção de tecnologias disruptivas. Espera-se que o destino final desses caminhos seja a existência de produtos e serviços mais justos, transparentes e com valor amplificado para todas as partes envolvidas, contrapondo-se ao funcionamento do mercado atual, em que os consumidores têm seus dados usados para maximizar o lucro das companhias (GLOBO, 2022).

## 1.2 OBJETIVO GERAL

O presente trabalho de conclusão de curso tem o objetivo final de apresentar uma prova de conceito que, sob contextos bem delimitados, mostrará a viabilidade do uso de *Smart Contracts* baseados em *Blockchain* para seguros parametrizados descentralizados, concentrando-se no setor aéreo.

## 1.3 OBJETIVOS ESPECÍFICOS

a) Analisar o mercado de seguros, enfatizando-se os seguros parametrizados;
b) Aplicar os conceitos fundamentados na revisão bibliográfica para a construção de uma aplicação que permite a contratação de um seguro parametrizado contra voos atrasados ou cancelados em contextos delimitados;
c) Apresentar conclusões qualitativas a respeito da aplicação desenvolvida.

## 1.4 TRABALHOS RELACIONADOS

Apesar do presente trabalho poder ser considerado vanguardista no Brasil, existem elementos que se correlacionam com outros projetos acadêmicos e da indústria. Dentre os que poderiam ser citados, o que mais se relaciona é o *Etherisc*.

*Etherisc* pode ser considerado um ecossistema para aplicações de seguros descentralizados. O grande objetivo deste ecossistema é permitir o uso da tecnologia *Blockchain* para uma construção de produtos de seguro mais eficientes, justos e transparentes (MUSSENBROCK; KARPISCHEK, s.d.). Para atingir tal objetivo, é feito o uso da tecnologia *Blockchain* e da rede *Ethereum*.

O projeto se relaciona com o presente trabalho em dois pontos: uso da tecnologia *Blockchain* e aplicação na indústria de seguros. Contudo, diferente da prova de conceito que será apresentada como resultado deste trabalho de conclusão de curso, A *Etherisc* não se posiciona como uma companhia de seguros descentralizada, mas como uma plataforma que possibilita que qualquer pessoa construa um produto de seguro de maneira coletiva.

# 2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem a intenção de embasar o presente trabalho, apresentando a fundamentação teórica por trás da prova de conceito desenvolvida.

## 2.1 SISTEMAS DESCENTRALIZADOS

Como discutido por Bodó e Giannopoulou (2019), na história da *Internet*, nota-se uma grande esperança de que tecnologias de código aberto, descentralizadas e distribuídas poderão dar origem a revolucionários modelos sociais, políticos e econômicos.

Neste contexto, o *Blockchain*, conceito definido por um autor desconhecido sob o pseudônimo Satoshi Nakamoto, fortificou ainda mais a crença de diversos grupos da sociedade de que uma tecnologia baseada em descentralização poderia solucionar uma variedade de problemas e desafios atuais, como: governança do sistema financeiro global, o poder excessivo de plataformas *online* e a maneira como é mantido o registro de propriedade (BODÓ; GIANNOPOULOU, 2019).

Portanto, é interessante apresentar e analisar ideias e projetos que possibilitam a implementação de aplicações descentralizadas, visando entender o nível de maturidade e o possível impacto na vida das pessoas.

### 2.1.1 *Open Innovation*

Com o grande volume de dados disponível atualmente, há a famosa concepção de que os dados são o novo petróleo (HUMBY, 2006). Contudo, analisando-se friamente, nota-se que apenas possuir uma grande quantidade de informação não levará a lugar algum. Portanto, para que sejam úteis, os dados são extremamente dependentes de um ecossistema. Nesse contexto, surge o termo *Open Innovation* ou, em tradução livre, Inovação Aberta.

*Open Innovation*, como define Chesbrough (2003), é um paradigma que presume que as empresas, à medida que elas buscam avançar a sua tecnologia, deveriam usar tanto ideias externas quanto ideias internas, assim como caminhos internos e externos para o mercado. Portanto, *Open Innovation* combina recursos internos e externos em arquiteturas e sistemas, cujos requerimentos são definidos por um modelo de negócio. Em outras palavras, trata-se de um paradigma em que há o compartilhamento livre de informações com fontes internas e externas para resolver problemas e criar soluções.

É possível enumerar diversas manifestações do paradigma *Open Innovation*, como: *Open Source*, *Open Banking*, *Open Insurance* etc. Algumas soluções muito usadas no dia a dia das pessoas são, por sua vez, derivadas dessas manifestações. O Linux, por exemplo, é um sistema operacional *Open Source* e gratuito. *Ethereum*, uma plataforma *Blockchain* emergente, também é *Open Source*. A quantidade de exemplos possíveis é vasta.

### 2.1.2 Cadeia de Blocos Criptograficamente Segura

Apesar do termo *Blockchain* ter sido cunhado junto do surgimento do *Bitcoin*, os fundamentos que o possibilitaram são mais antigos. Em um artigo escrito por Haber e Stornetta (1990), eles propuseram um processo computacionalmente prático para possibilitar que registros de data e hora de um documento fossem invioláveis e não retroativos. Tratava-se, por exemplo, de uma proposta de solução extremamente viável para a manutenção de propriedades intelectuais, visto que é crucial verificar a data em que um inventor escreveu sua ideia patenteável para eventuais reivindicações.

O princípio básico da solução proposta é a existência de um cofre digital, em que o documento que precisa ter a marca temporal é transmitido para o Serviço de *Time-Stamping* (*TTS*). Então, o *TTS* registrará a data e horário de recebimento do documento e salvará uma cópia por segurança. Sempre que a integridade do documento for posta à prova deverá ser feita uma comparação com a cópia salva pelo *TTS* para verificar qualquer tipo de adulteração. A existência do cofre digital é considerada um princípio e não a solução completa por apresentar os seguintes problemas: falta de privacidade, alta demanda de largura de banda e armazenamento, incompetência operacional na transmissão do documento e falta de confiabilidade (HABER; STORNETTA, 1990).

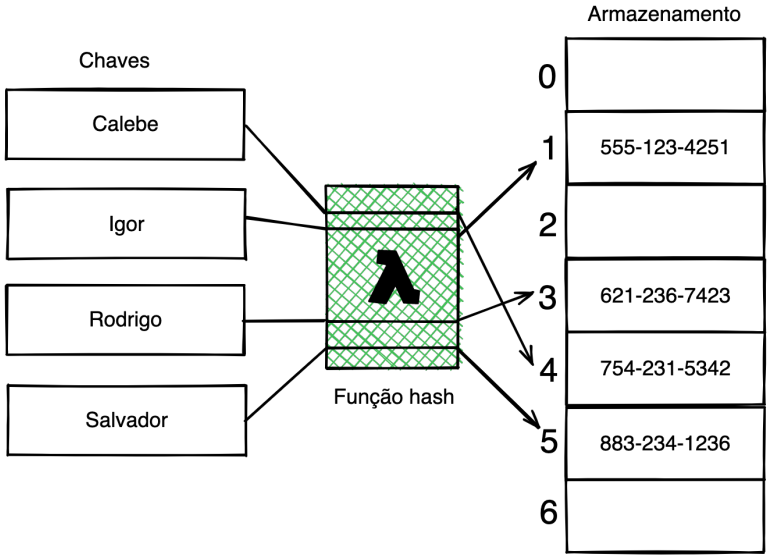
Para lidar com os problemas supramencionados, é proposta a inclusão de novos conceitos que darão mais robustez à solução. Cria-se, então, em tradução livre, um Serviço Confiável de Marca Temporal, que consiste do uso de funções *Hash*, esquemas de assinatura e geradores de números pseudoaleatórios (HABER; STORNETTA, 1990).

#### *2.1.2.1 Funções Hash*

O trabalho propõe que seja enviado apenas o *Hash* do documento para ser feita a marca temporal ao invés de enviar o documento inteiro. Tal proposta permite contornar os problemas de privacidade e segurança, além da alta demanda de largura de banda e armazenamento (HABER;

STORNETTA, 1990). A figura 1 ilustra, de forma simplificada, o uso de funções *Hash* para criptografar um conteúdo. Uma função *Hash* é uma função na qual você insere um dado e ela retorna um número. Portanto, é uma função que mapeia dados e os relaciona a números (BHARGAVA, 2016).

Figura 1 – Exemplo de Encriptação *Hash*




Fonte: o Autor

#### *2.1.2.2 Esquema de Assinatura*

Para evitar o problema de incompetência operacional e proporcionar melhorias no armazenamento, há a adoção de um esquema de assinatura. Quando o *TTS* recebe o *Hash*, ele anexa a data e hora e, então, é feita uma assinatura digital no documento antes dele ser enviado novamente para a parte interessada. Incluir a assinatura no documento garante, portanto, que o *TTS* processou a requisição, que o *Hash* foi corretamente recebido e que o tempo correto foi incluso (HABER; STORNETTA, 1990).

#### *2.1.2.3 Linking e Segurança Distribuída*

A inclusão de função *Hash* e do esquema de assinatura certamente aumentam a eficácia do *TTS*, contudo, só se estiver sendo considerada a hipótese de que o *TTS* é seguro. Considerando-se que não seja, tem-se a necessidade, por conseguinte, de adicionar mais uma camada à solução para ela ficar completa (HABER; STORNETTA, 1990).

*Linking*, que é a primeira abordagem proposta para a garantia de um *TTS* confiável, consiste na inclusão dos bits da sequência anterior de requisições do cliente no certificado assinado (HABER; STORNETTA, 1990).

Segurança Distribuída, que é a segunda abordagem, considera que cada cliente possui um esquema de assinatura seguro e um gerador de número pseudoaleatório padronizado. Tal abordagem considera que o número de clientes desonestos sempre será menor que o número de clientes honestos, o que impossibilitará fraudar o sistema e tirará a necessidade de uma autoridade central, visto que para fraudar seria necessário deter um controle majoritário do sistema (HABER; STORNETTA, 1990).

Por fim, os autores concluem que a união das duas abordagens torna viável a adição de marcas temporais em documentos digitais. O conteúdo discutido no artigo foi extremamente importante para o surgimento da tecnologia *Blockchain* e, consequentemente, do *Bitcoin*.

### 2.1.3 *Bitcoin*

O *Bitcoin* surgiu como uma solução para o problema de gasto duplo, que é quando é possível gastar a mesma moeda mais de uma vez (NAKAMOTO, 2008). Segundo Nakamoto (2008), *Bitcoin*, que é uma moeda eletrônica, pode ser definida como uma cadeia de assinaturas digitais em que, como ilustra a figura 2, cada proprietário transfere a moeda para o próximo por meio de uma assinatura digital de *Hash* da operação anterior e pela chave pública do próximo dono, adicionando-se esses dados ao fim da moeda. Então, um sacador pode verificar a cadeia de propriedade das assinaturas.

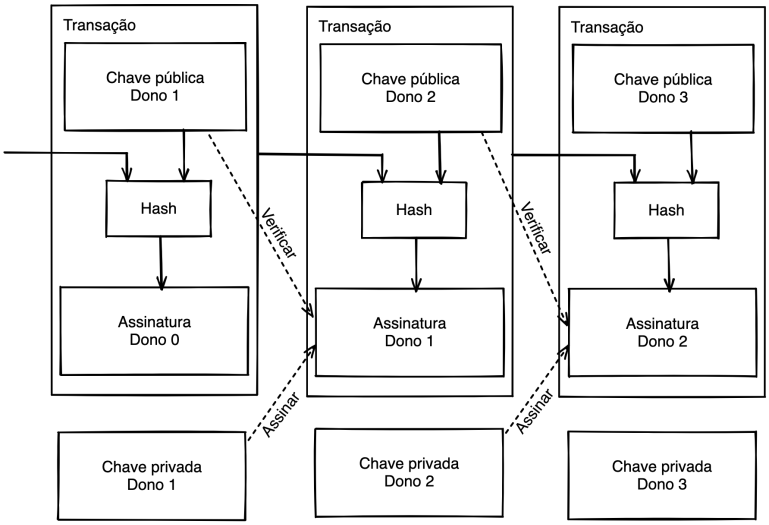
Contudo, o que foi supracitado ainda não resolve o problema de gasto duplo. A solução simples consiste em inserir uma autoridade central, mas esse não é o objetivo. Um caminho viável para que seja possível validar que os proprietários anteriores não assinaram qualquer transação anterior sem uma governança centralizada é realizar o anúncio das transações publicamente e incluir um sistema em que os participantes concordem com a ordem das transações recebidas em um histórico único, que pode ser chamado de livro-razão. Portanto, é necessário que a maioria dos participantes reconheça que uma dada transação está sendo recebida pela primeira vez (NAKAMOTO, 2008).

A solução proposta para resolver o problema de gasto duplo é a criação de uma rede *Peer-to-peer*[1] que usa um algoritmo de consenso chamado *Proof of Work* (prova de trabalho, em

[1] *Blockchain* é uma rede *Peer-to-peer*.

tradução livre). Tal algoritmo é responsável por gravar um histórico público que rapidamente se torna computacionalmente impraticável para um atacante fraudar, visto que é necessário deter poder majoritário de CPU, isto é, mais de 50% do poder computacional da rede *Blockchain* inteira (NAKAMOTO, 2008).

Figura 2 – Transação de *Bitcoin*




Fonte: Nakamoto, 2008

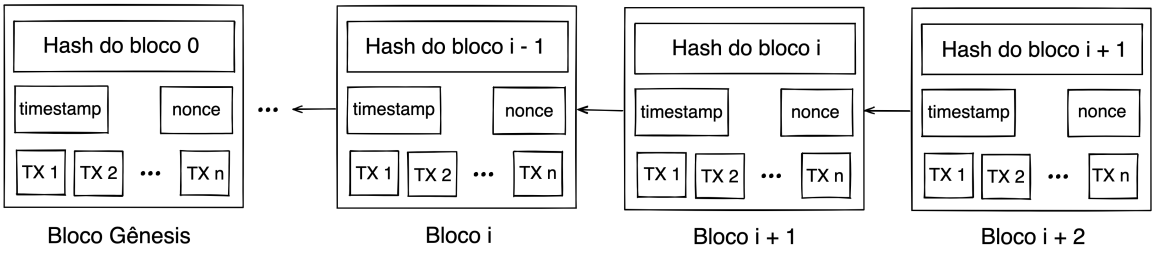
### 2.1.4 *Blockchain*

*Blockchain* é uma sequência de blocos em que, como um livro-razão, é armazenada uma lista completa dos registros de transações (ZHENG et al., 2016 apud CHUEN, 2015). Conforme ilustra a figura 3, cada bloco aponta para o anterior imediato por meio de uma referência, que é um valor *Hash* do bloco anterior (chamado de pai). A origem do *Blockchain* tem o bloco gênesis, sendo que este não possui bloco pai (ZHENG et al., 2016).

Um bloco se caracteriza pela cabeça e pelo corpo. A figura 4 mostra a composição de um bloco. De acordo com Zheng et al. (2016), pode-se destacar os seguintes elementos da cabeça:

a) Versão do bloco: indica qual conjunto de validação de regras seguir;
b) *Hash* do bloco pai: um código *Hash* de 256 *bits* que aponta para o bloco imediatamente anterior;
c) *Hash* da raiz da árvore de Merkle: o código *Hash* de todas as transações no bloco;
d) *Timestamp* (Marca Temporal, em tradução livre): o *timestamp* atual desde 01 de Janeiro de 1970 (UTC);

Figura 3 – Exemplo de Arquitetura *Blockchain*



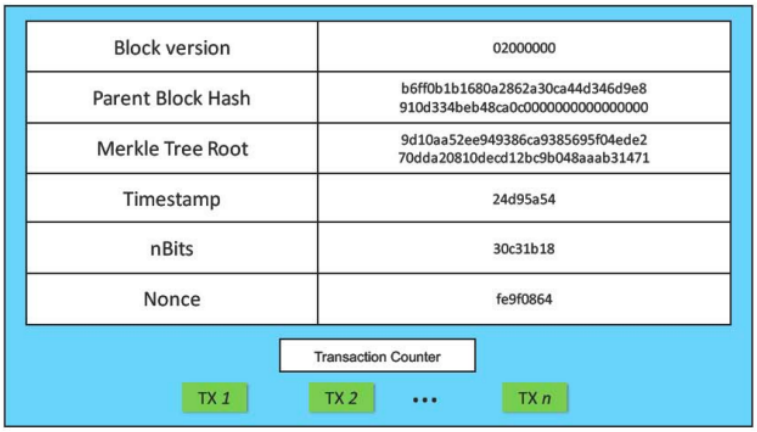

Fonte: Zheng et al., 2016

e) *nBits*: próximo alvo *Hash* de forma compactada.

f) *Nonce*: um campo de 4 *bytes*, que, comumente, começa em zero e cresce em cada cálculo *Hash*

Ainda, tendo-se a figura 4 como referência, nota-se a presença do corpo, que é composto por um contador de transações e pelas transações (ZHENG et al., 2016).

Figura 4 – Estrutura de um Bloco




Fonte: Zheng et al., 2016

#### *2.1.4.1 Assinatura Digital*

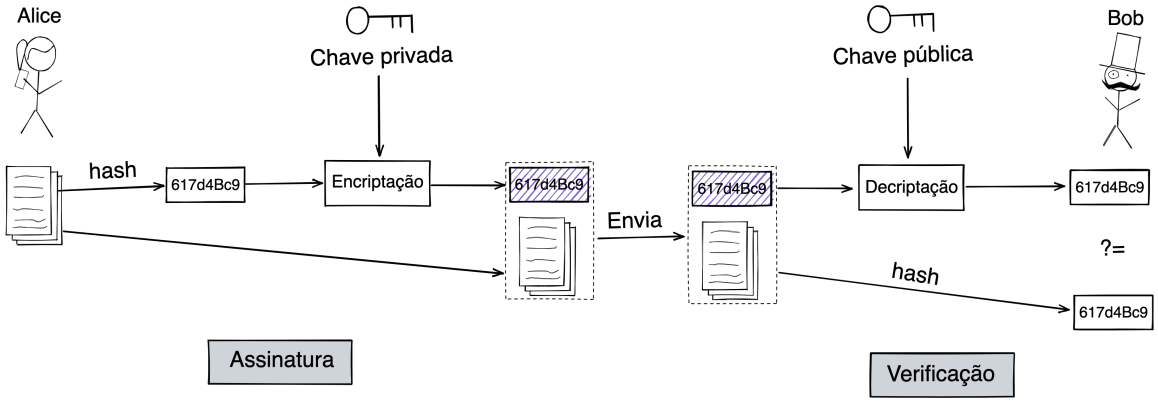
Para que uma transação seja autenticada, faz-se o uso de um mecanismo criptográfico assimétrico. Cada usuário da rede possui uma chave pública e uma privada. Enquanto a chave privada é responsável por assinar as transações, a chave pública, que é visível para todos, permite acessá-las, em virtude do compartilhamento dessas transações por toda a rede. A figura 5

demonstra o funcionamento das duas fases (assinatura e verificação) de uma assinatura digital (ZHENG et al., 2016).

#### *2.1.4.2 Características chaves do Blockchain*

Por fim, pode-se elencar as principais características que norteiam o *Blockchain*: descentralização, persistência, anonimidade e auditabilidade (ZHENG et al., 2016). Tais características são, portanto, o que permitiu com que Nakamoto resolvesse o problema de gasto duplo e promovesse uma tecnologia poderosa com enorme potencial para diversos setores da indústria.

Figura 5 – Assinatura Digital Usada no *Blockchain*




Fonte: Zheng et al., 2016

### 2.1.5 Algoritmos de Consenso

Na subseção 2.1.3 é dito que o *Bitcoin* surgiu como uma solução para o problema de gasto duplo de maneira descentralizada. Contudo, o emprego da tecnologia *Blockchain* implica, necessariamente, na solução de um segundo problema: Problema dos Generais Bizantinos (*Byzantine Generals Problem*).

Como descreve Mingxiao et al. (2017), o Problema dos Generais Bizantinos, que discute como garantir a confiabilidade e não adulteração dos enlaces de informação, está presente em sistemas distribuídos. O *Blockchain*, que é um sistema distribuído baseado em nós, resolve a problemática de gasto duplo, mas alguns nós da rede podem sofrer ataques maliciosos, resultando em mudanças nos conteúdos transmitidos. Dada a necessidade de distinguir informações adulteradas das consistentes, é feito o uso de algoritmos de consenso.

Existe uma vasta quantidade de algoritmos de consenso que visam resolver o Problema dos Generais Bizantinos. No artigo escrito por Bamakan, Motavali e Babaei Bondarti (2020), tem-se a especificação de 12 algoritmos. Este trabalho, por sua vez, concentrará-se na explicação de dois dos mais populares: *Proof of Work* e *Proof of Stake*.

#### *2.1.5.1 Proof of Work (PoW)*

Apoiando-se do trabalho de Back et al. (2002), o uso de *Proof of Work* foi proposto por Nakamoto (2008). Conforme ele explica, o algoritmo busca por um valor que, quando calculado o *Hash SHA*-256[2], comece com uma determinada quantidade de *bits* zero. A dificuldade de encontrar esse valor cresce exponencialmente de acordo com a quantidade de bits zero requerida. A verificação, por outro lado, é muito simples.

Segundo Mingxiao et al. (2017), os passos para encontrar esse valor especial, que é tratado como a solução de um problema matemático, são os seguintes:

a) Obter a dificuldade: a dificuldade é ajustada de acordo com a taxa de *Hash* de toda rede *Blockchain*;

b) Coletar as transações: coleta-se todas as transações pendentes na rede depois da produção do último bloco. Calcula-se a raiz de Merkle dessas transações coletadas e insere o código *Hash* de 256 *bits* do bloco anterior, o valor de *Hash* atual, o número *Nonce* aleatório e outras informações pertinentes no número de versão de bloco;

c) Calcular: Percorre o *Nonce* de zero a $2\hat{}32$ e calcula o dobro do valor *Hash SHA-256* do passo b. Se o valor é menor ou igual ao valor alvo, o bloco pode ser transmitido. Então, após a verificação de outros nós a contabilidade é concluída;

d) Reiniciar: se o valor de *Hash* não for descoberto em um tempo determinado, o passo b é repetido. Agora, caso qualquer outro nó complete o cálculo, volta-se para o passo a.

Cada bloco da rede *Blockchain* proposta por Nakamoto (2008) é originado da solução desse desafio matemático. Por ser extremamente custoso computacionalmente chegar-se à solução, há uma recompensa para o minerador[3] responsável pela criação do bloco (MINGXIAO et al., 2017).

[2] O algoritmo *SHA*-256 é uma maneira de codificar informações.

[3] Os mineradores são responsáveis por garantir a veracidade das informações. A mineração é feita por um processo computacional complexo.

É possível notar que *Proof of Work* usa a carga de trabalho como premissa da confiabilidade da rede. Sendo esta a premissa, para que seja viável fraudá-la, deve-se possuir mais de 50% do poder computacional dela. Considerando-se que a maioria do poder computacional da rede é controlado por nós honestos, tornará-se, probabilisticamente, cada vez mais difícil uma fraude, visto que a rede honesta crescerá mais rapidamente em comparação àquela que houve uma tentativa de ataque (NAKAMOTO, 2008).

#### *2.1.5.2 Proof of Stake (PoS)*

*PoS* é um algoritmo que visa resolver o problema de elevado poder computacional exigido no *PoW*. No *PoS*, atribui-se o conceito de idade à moeda digital. O valor da moeda multiplicado pelo tempo que ela foi criada constitui a sua idade. Então, quanto mais tempo uma moeda é mantida em um nó da rede, maiores são os direitos de seu detentor (MINGXIAO et al., 2017).

Na rede que adota o *PoS*, é estimulada a detenção da moeda por um longo período. Como os direitos estão diretamente atrelados à idade da moeda, é necessário que os atacantes possuam uma grande quantidade de moedas por um longo período (MINGXIAO et al., 2017). Como tudo, a adoção deste algoritmo tem seus pontos fortes e fracos. Se por um lado ela aumenta a economia energética da rede, por outro ela aumenta a centralização de poder (BAMAKAN; MOTAVALI; BABAEI BONDARTI, 2020).

#### *2.1.5.3 Comparativos Entre PoW e PoS*

Existem diferentes categorizações de *Blockchain*: público, consorciado e privado (BAMAKAN; MOTAVALI; BABAEI BONDARTI, 2020). Cada categoria, por conseguinte, pode ter um algoritmo de consenso que se adéqua melhor. Tendo como enfoque os dois que foram mais aprofundados neste trabalho, pode-se fazer algumas análises úteis para definir suas aplicabilidades.

Conforme pode ser visto na tabela 2, as maiores diferenças performáticas dos dois algoritmos são a velocidade de verificação e taxa de transferência. A tabela 3, por sua vez, faz uma boa sumarização das diferentes características deles.

Tabela 2 – Comparação de Desempenho Entre *PoW* e *PoS*

| **Características** | *PoW* | *PoS* |
|---|---|---|
| Tolerância de falha Bizantina | 50% | 50% |
| Tolerância de falha de colisão | 50% | 50% |
| Velocidade de verificação | >100s | <100s |
| Taxa de transferência (TPS) | <100 | <1000 |
| Escalabilidade | alta | alta |

Fonte: Mingxiao et al., 2017

Tabela 3 – Comparação das Características Entre *PoW* e *PoS*

| | *PoW* | *PoS* |
|---|---|---|
| **Objetivo Arquitetural** | À prova de sibilas | Eficiência energética |
| **Nível de Descentralização** | Descentralização | Semi-centralizado |
| **Modelo de Permissão** | Sem permissão | Sem permissão |
| **Tipo de Verificação** | Trabalho | Detenção |
| **Eficiência Energética** | Não | Sim |
| **Escalabilidade** | Forte | Forte |
| **Ataque da Maioria (51%)** | Vulnerável | Vulnerável |
| **Ataque de Gasto Duplo** | Vulnerável | Difícil |
| **Dependência de *Hardware*** | Sim | Não |
| **Velocidade** | Lento | Veloz |

Fonte: Bamakan, Motavali e Babaei Bondarti, 2020

### 2.1.6 *Smart Contracts*

Como define Szabo (1997), o contrato é um conjunto de promessas acordadas que formaliza um relacionamento. Um contrato é, portanto, uma peça fundamental para o mercado econômico.

Com o avanço computacional permitindo a execução de algoritmos que antes eram extremamente inviáveis e com os avanços na rede, ainda, como propõe Szabo (1997), tem-se um espaço viável para novas maneiras de formalizar relações.

O conceito de *Smart Contract*, definido pioneiramente por Szabo (1997), defende a ideia de que muitos tipos de cláusulas contratuais podem ser embarcadas em um *software* ou *hardware*, tornando, assim, a quebra de contrato cara e complexa para as partes envolvidas. Um bom exemplo para isso é uma *vending machine*, que é considerada uma ancestral do *Smart Contract*. Em uma *vending machine*, os que detêm moedas podem interagir com o mecanismo

que devolverá um recurso de interesse. Uma vez que a quantidade ideal de moeda é inserida, uma saída é garantida[4] e ambas as partes têm suas cláusulas cumpridas.

O exemplo acima levanta uma hipótese muito forte. Assim como a *vending machine* remove a necessidade de um vendedor cuidando dela, isto é, de um intermediário, os *Smart Contracts* podem substituir intermediários em muitas indústrias, visto que se trata de um contrato auto-executável com regras bem definidas (COHN; WEST; PARKER, 2016).

#### *2.1.6.1 Smart Contracts: Uma Abordagem Blockchain*

É importante ressaltar que conceito de *Smart Contract* não está intrinsecamente relacionado ao *Blockchain*. Na verdade, refere-se mais à ideia de um tipo de transação auto-executável (COHN; WEST; PARKER, 2016).

Agora, abordando-se sob a perspectiva do *Blockchain*, o *Smart Contract* pode ser visto como um contrato entre duas ou mais partes que será armazenado e executado no *Blockchain* sem uma intervenção humana. Ou seja, dependerá apenas do código e dos parâmetros que foram definidos nele (COHN; WEST; PARKER, 2016 apud CLACK; BAKSHI; BRAINE, 2016).

O código-fonte a seguir ilustra uma *vending machine* vista como um *Smart Contract*.

```
1
2  pragma solidity 0.8.7;
3
4  contract VendingMachine {
5    // Declara as variaveis de estado do contrato
6    address public dono;
7    mapping (address => uint) public saldoCupcakes;
8
9    // Quando o contrato vendingMachine for implementado na rede:
10   // 1. define o endereco de deploy como dono do contrato
11   // 2. define a quantidade de cupcakes do smart contract como 100
12   constructor() {
13     dono = msg.sender;
14     saldoCupcakes[address(this)] = 100;
15   }
16
17   // Permite que o dono do contrato aumente o saldo de cupcakes do smart
       contract
```

[4]Considerando uma situação ideal em que não haja falha no mecanismo nem tentativa de violação física do equipamento.

```
18    function refil(uint quantidade) public {
19      require(msg.sender == dono, "Somente o dono pode fazer a reposicao."
          );
20      saldoCupcakes[address[this]] += quantidade;
21    }
22
23    // Permite qualquer pessoa adquirir cupcakes
24    function purchase(uint quantidade) public payable {
25      require(msg.value >= quantidade * 1 ether, "Voce deve pagar pelo
          menos 1 ETH por cupcake");
26      require(saldoCupcakes[address(this)] >= quantidade, "Nao ha cupcakes
           disponiveis no estoque para finalizar esta transacao");
27      saldoCupcakes[address(this)] -= quantidade
28      saldoCupcakes[msg.sender] += quantidade
29    }
30  }
```

Código-fonte 2.1 – *Smart Contract* de uma *Vending Machine* Escrito em *Solidity*

#### *2.1.6.2 Ethereum*

Enquanto o *Bitcoin* foi desenhando com a finalidade de ser um sistema de pagamento *Peer-to-peer*, a *Ethereum* foi desenhada para ser mais do que um sistema de pagamentos. Segundo Buterin (2013), o objetivo foi possibilitar a construção de aplicações descentralizadas por meio de um protocolo alternativo ao *Bitcoin*. *Ethereum*, portanto, representa um *Blockchain* com uma linguagem de programação embutida (VUJIČIĆ; JAGODIĆ; RANĐIĆ, 2018).

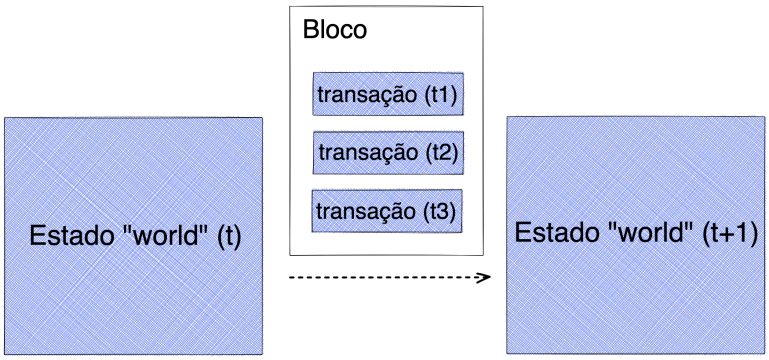
Sob uma perspectiva geral, *Ethereum* é uma máquina de estado baseada em transação. Ou seja, existe o estado inicial e, à medida que são executadas transações, chega-se em algum estado final, que pode conter qualquer tipo de informação cuja representação computacional é possível (WOOD et al., 2014).

Por representar a transição entre dois estados, a transação deve ser considerada válida. A figura 6 ilustra uma transação e a figura 7 ilustra o encapsulamento das transações em um bloco, devido à necessidade de assegurar a sincronização do estado e a precisão do histórico de transações (WOOD et al., 2014).

Figura 6 – Transação Entre Estados


Fonte: o Autor (com base na documentação da Ethereum, 2022)

Figura 7 – Bloco com Transações Encapsuladas




Fonte: o Autor (com base na documentação da Ethereum, 2022)

Na *Ethereum*, um bloco é levemente diferente do que foi discutido na subseção 2.1.4. A maior diferença, comparada ao *Blockchain* do *Bitcoin*, é que o bloco *Ethereum* inclui a lista de transações e o estado mais recente (VUJIČIĆ; JAGODIĆ; RANĐIĆ, 2018).

#### *2.1.6.3 Smart Contract e Ethereum*

Do ponto de vista de *Ethereum*, *Smart Contract* é um programa rodando na sua *Blockchain*, que contém funções, estados e reside em um endereço específico (ETHEREUM, 2022).

*Ethereum* possui dois tipos de conta: externamente controlada por chaves privadas e *Smart Contract*. Portanto, sendo uma conta, *Smart Contracts* também possuem saldo e podem transacionar pela rede, mas sem ser controlada por um usuário. Apesar de não ser controlada por um usuário, ela possibilita a interação por meio da determinação de regras no código. Um fato importante é que as interações envolvendo *Smart Contracts* são irreversíveis (BUTERIN, 2013).

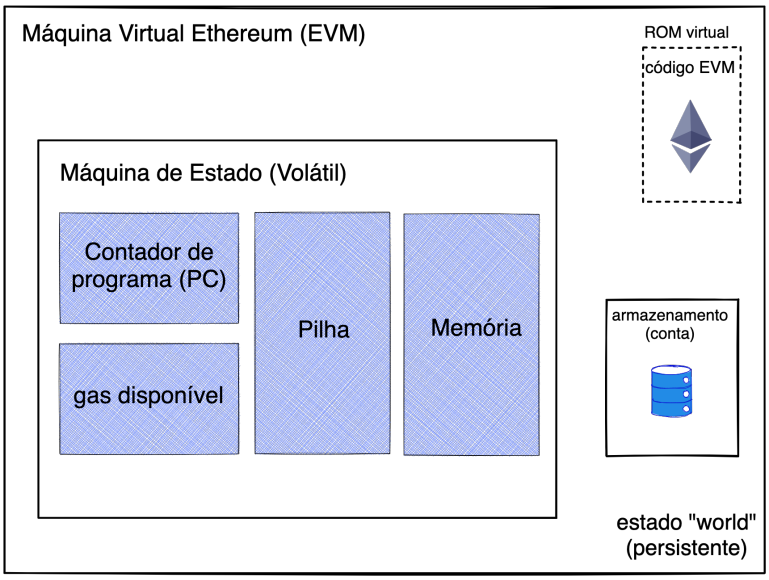
#### *2.1.6.4 Máquina Virtual Ethereum (EVM)*

A *EVM*, que é o ambiente na qual as contas e os *Smart Contracts* vivem, possui uma arquitetura baseada em pilha, conforme ilustra a figura 8. A *EVM* não segue a clássica arquitetura computacional de von Neumann (WOOD et al., 2014).

Conforme descreve a documentação da Ethereum (2022), A *EVM* é executada como uma máquina de pilha com a profundidade de 1024 itens, sendo cada item uma palavra de 256 *bits*. Durante a execução, é mantida uma memória transiente sem persistência entre as transações. Os *Smart Contracts*, por sua vez, possuem uma *Merkle Patricia Trie* associada aos seus endereços e com parte do estado global. Uma *Merkle Patricia Trie* é uma versão modificada da *Merkle Patricia Tree*, responsável por prover uma estrutura de dados persistente para mapear dados binários de comprimentos arbitrários. É, portanto, uma estrutura que provê um valor único para mapear um conjunto de chave-valor (WOOD et al., 2014).

Para cada operação na rede *Ethereum*, há um custo computacional associado. *Gas* (combustível, em tradução livre) é o termo utilizado para contabilizar a quantidade do esforço necessário para executar uma operação. Portanto, cada transação tem uma *fee* (taxa, em tradução livre) que visa reembolsar os mineradores pelo trabalho executado. *Gas fee*, o custo da transação, é pago em *ether*, a criptomoeda nativa da *Ethereum* (ETHEREUM, 2022).

Figura 8 – Arquitetura da *EVM*




Fonte: o Autor (com base na documentação da Ethereum, 2022)

## 2.2 ARQUITETURA *WEB* DE APLICAÇÕES DESCENTRALIZADAS

A arquitetura de *software* é parte essencial do desenvolvimento de software. Afinal, como defende Mead et al. (2004), é onde as ideias e ambiguidades são traduzidas e transformadas em realidade.

Quando o olhar é voltado à arquitetura de sistemas *web*, apoiando-se do trabalho de Voshmgir (2020), é possível elencar três gerações distintas:

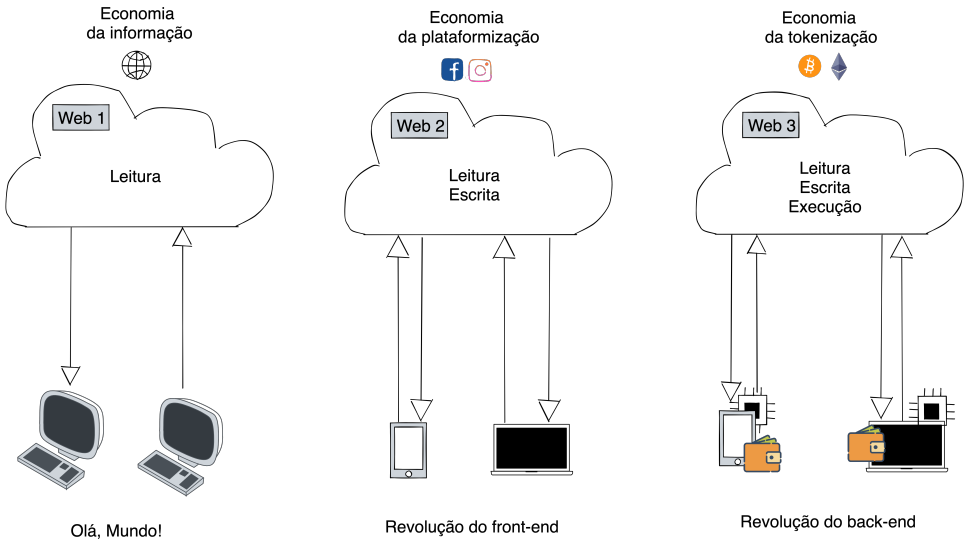
a) web1: o foco estava na leitura. O acesso à informação se tornou mais favorável;

b) web2: a *Internet* ficou mais madura, abrindo espaço para a escrita (além da leitura). A web2 pode ser considerada, também, a revolução do *front-end*;

c) web3: se a web2 é a revolução do *front-end*, a web3 é a revolução do *back-end*. Aqui, apesar do *front-end* permanecer como era na web2, as estruturas de dados do *back-end* foram modificadas.

A figura 9 faz um breve sumário das três gerações supracitadas. Sob o ponto de vista econômico, Voshmgir (2020) as diferenciam como: economia da informação, economia de plataforma e economia de *tokenização*.

Figura 9 – As Três Gerações da *WEB*




Fonte: Voshmgir, 2020

### 2.2.1 O Caminho Para Uma Arquitetura Descentralizada

De acordo com o artigo de Zarrin et al. (2021), há dois tipos de redes descentralizadas que viabilizam o ideal de *Internet* descentralizada. O primeiro consiste de uma rede com arquitetura completamente descentralizada, em que a confiança e controle é espalhada entre usuários anônimos. Uma grande desvantagem atrelada a esse modelo é ter que abrir mão das tecnologias convenientes desenvolvidas na web2.

O outro método é com o uso de uma rede distribuída, como é o caso do *Blockchain*. Neste modelo, a rede distribuída garante a interconexão e co-dependência entre os participantes. Uma grande vantagem desse modelo é o oferecimento da possibilidade de converter sistemas centralizados para descentralizados (ZARRIN et al., 2021).

Portanto, adotando-se *Blockchain*, o caminho para a descentralização se torna viável.

### 2.2.2 *Web3*

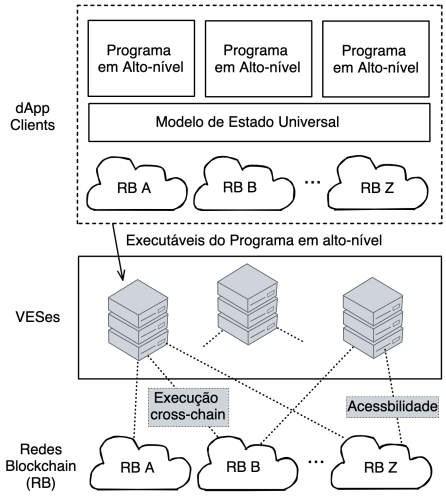
Apesar de não ser a única forma de entender *web3*, como define Liu et al. (2021), ela pode ser vista como uma nova era da computação em que a parte computacional crítica das aplicações é verificável. Portanto, seguindo tal definição e ainda de acordo com Liu et al. (2021), existem três princípios facilitadores para a *web3*:

a) *Blockchain* com a capacidade de executar *Smart Contracts* de maneira performática e confiável;
b) Inclusão de plataformas federadas ou centralizadas para compensar as funcionalidades que são inviáveis dentro de uma rede *Blockchain*;
c) Uma plataforma segura de interoperabilidade para conseguir lidar com a distribuição dos estados de uma aplicação *web3* publicados por sistemas diferentes e isolados.

O resultado arquitetural do que foi enumerado acima, com ênfase no item c, pode ser visto na figura 10. Pode-se destacar três elementos cruciais da arquitetura:

a) DAPPs: podem ser vistos como a forma alto-nível de se relacionar com as redes *Blockchain*;
b) Sistemas de Execução Verificáveis (VESes): têm o papel de transformar as interações de alto nível em algo entendível para a rede *Blockchain*;
c) Redes *Blockchain*: registram e validam as transações ocorridas.

Figura 10 – Arquitetura de uma Plataforma Segura de Interoperabilidade




Fonte: o Autor (com base em Liu et al., 2021)

### 2.2.3 *Blockchain Oracle*

Uma das limitações dos *Smart Contracts* baseados em *Blockchain* é que eles não podem executar dados externos à rede. Tal limitação pode impactar significativamente na construção de modelos de negócios, visto que eles podem depender de informações externas à rede *Blockchain*.

Visando contornar a problemática apresentada acima, tem-se a ideia de *Blockchain oracles* (oráculos *Blockchain*, em tradução livre)[5], que são serviços terceirizados para prover *Smart Contracts* com informações externas à rede *Blockchain* que ele está inserido (BENIICHE, 2020).

Existem muitos tipos de oráculos e é importante deixar claro que um oráculo *Blockchain* não é o dado propriamente, mas sim uma forma de consultar, verificar e autenticar as informações externas. Alguns tipos diferentes de oráculos que podem ser citados são: oráculos por *software*, oráculos por *hardware*, oráculos humanos, oráculos computacionais, oráculos de entrada/saída, oráculos baseados em especificidades contratuais e oráculos baseados em consenso (BENIICHE, 2020).

Ainda, segundo Beniiche (2020), é possível definir os oráculos como sendo centralizados ou descentralizados. Um oráculo centralizado não é uma boa prática, em virtude de ser controlado por uma única entidade e ser o único provedor de informação para o *Smart Contract*. Apesar de

[5] *Oracles* e oráculos serão usados de maneira intercambiável ao longo do trabalho, mas têm o mesmo significado

não ser a melhor prática, por fins de facilidade de implementação, será a estratégia adotada para a prova de conceito proposta por este trabalho.

Em contrapartida, oráculos descentralizados, que podem ser categorizados como oráculos de consenso, aumentam a confiança da informação provisionada ao *Smart Contract* por não terem uma única fonte de verdade. Ou seja, são feitas várias consultas em diferentes fontes de informação para garantir o consenso dela.

Na indústria tem alguns projetos de *Blockchain* que possibilitam oráculos descentralizados. Os projetos de mais destaque são: *Provable* e *Chainlink*.

## 2.3 MERCADO DE SEGUROS

Como definido por Scherrer e Salahshor (2020), seguro é um produto financeiro na forma de contrato que protege um indivíduo contra perdas. No seu formato clássico, consiste do pagamento de um prêmio do segurado ao segurador. A quantidade a ser paga pelo segurado é definida por meio de um processo de *underwriting*, em que os riscos são sistematicamente medidos, a fim de estabelecer a quantidade monetária atreladas a eles.

O modelo de negócio da indústria de seguros é baseado em modelos de riscos. Há vários segurados que investem no mesmo produto. Diferentes perfis de segurados têm acesso a diferentes valores de riscos associados ao produto em que se tem o investimento. Como não é um cenário muito provável que todos os segurados vão sofrer dos eventos que possibilitam solicitar o prêmio estabelecido em contrato, tem-se o acúmulo de prêmio suficiente para que seja um modelo de negócio sustentável (SCHERRER; SALAHSHOR, 2020).

Por ainda ser uma indústria extremamente conservadora, com o suporte tecnológico, tem-se espaço para muita inovação. Como já discutido neste trabalho, manifestações da *Open Innovation* e plataformas *Blockchain* possibilitam repensar o modelo e propor mudanças que colocam o consumidor no centro.

### 2.3.1 Seguro Paramétrico

Segundo Ibarra e Securities (2010), o seguro paramétrico pode ser definido como um contrato de seguro em que o pagamento de sinistro[6] é determinado por uma observação geológica ou meteorológica ou, ainda, por algum índice.

[6] Sinistro é a ocorrência de todo evento coberto pelo seguro contratado.

Outro aspecto importante do seguro paramétrico é a sua forma de pagamento. Como presente no artigo de Cohn, West e Parker (2016), os pagamentos não são determinados por um perito de sinistros que avalia os danos, mas sim com base em medidas objetivas, como, por exemplo, na magnitude de um evento climático ou se um voo atrasou ou não.

Nota-se, por conseguinte, que o seguro paramétrico depende de indicadores objetivos, uma vez que possui a premissa de não ter alguém responsável por avaliar os danos. Atingiu-se o parâmetro especificado, recebe-se o prêmio. Uma consequência, portanto, é que o seguro paramétrico pode diminuir o tempo de pagamento de meses para intervalos bem menores (COHN; WEST; PARKER, 2016).

Por se tratar de um produto baseado em parâmetros, torna-se extremamente propício para o uso de *Smart Contracts*, que permitirá otimizar as complexidades do negócio, desempenho e gerenciamento, além do ganho de transparência e confiabilidade (COHN; WEST; PARKER, 2016).

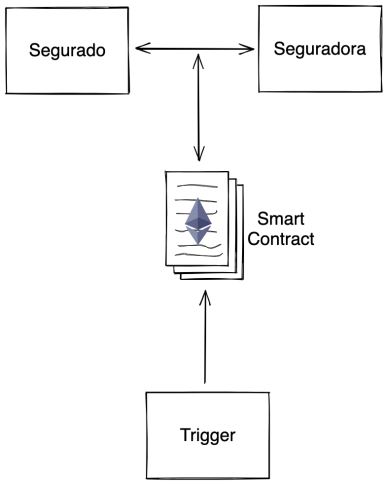
A figura 11 ilustra um fluxograma simplificado de como pode ser estabelecido um seguro paramétrico. Neste modelo, o segurado e seguradora estabelece uma relação e definem os parâmetros por meio de um *Smart Contract*. Então, um *trigger*, responsável por informar o contrato se o parâmetro especificado foi ou não atingido é disparado, ocasionando em uma ação por parte do *Smart Contract*.

Nota-se que, a partir do estabelecimento do *Smart Contract*, perde-se a necessidade do segurado fazer o envio de sinistro, algo necessário em um modelo de seguro clássico. Ou seja, trata-se de uma relação dependente apenas da definição de um parâmetro, tornando-a extremamente simples, transparente e eficiente.

### 2.3.2 Outros Modelos possíveis: revolucionando a indústria como um todo

Apesar deste trabalho se concentrar na indústria de seguros paramétricos, existem movimentos que vão além e prometem causar revolução na indústria de seguros como um todo. Existe, por exemplo, o trabalho de Loukil et al. (2021) que propõe a criação de um sistema de seguro colaborativo baseado em *Blockchain*. Ou seja, existem propostas de projetos que visam competir com os modelos clássicos de seguro.

Figura 11 – Diagrama de Seguro Paramétrico Usando *Smart Contract*




Fonte: o Autor

# 3 METODOLOGIA

O presente trabalho de conclusão de curso pode ser tratado como uma pesquisa exploratória de abordagem qualitativa, em que há uma forte revisão bibliográfica para embasar a prova de conceito que será apresentada a seguir.

## 3.1 DELIMITAÇÃO DE CONTEXTO

Como descrito na seção 1.2, é objetivo deste trabalho de conclusão de curso demonstrar a viabilidade do uso de *Smart Contracts* baseados em *Blockchain* para a indústria de seguros paramétricos. A partir do que foi apresentado até aqui, nota-se o grande potencial de aplicação para diversos modelos[1]. Contudo, dada a finalidade e pela maior facilidade de implementação, a prova de conceito será construída sob a perspectiva de seguro para voos atrasados ou cancelados.

### 3.1.1 Formulação do Problema

Em um mundo cada vez mais líquido e sob a premissa de que tempo é dinheiro, um atraso ou cancelamento de voo pode ser potencialmente prejudicial para alguém, tendo em vista que pode resultar em gastos não previstos (alimentação, acomodação etc). Inclusive, existe seguro viagem para atrasos e cancelamentos de voo[2]. Os trâmites para o acionamento do seguro, entretanto, são bem burocráticos, envolvendo-se a necessidade de possuir diversos documentos que comprovem o direito. Outro ponto negativo é que raramente o reembolso é em um prazo interessante para o consumidor.

### 3.1.2 Proposta de Solução

A solução proposta por este trabalho consiste em considerar uma relação descentralizada entre segurado e seguradora. O segurado será considerado aquele que deseja ser ressarcido pelo atraso ou cancelamento de um determinado voo. A seguradora, por sua vez, será a responsável por arcar com o ressarcimento da recompensa definida em caso de atraso ou cancelamento do voo.

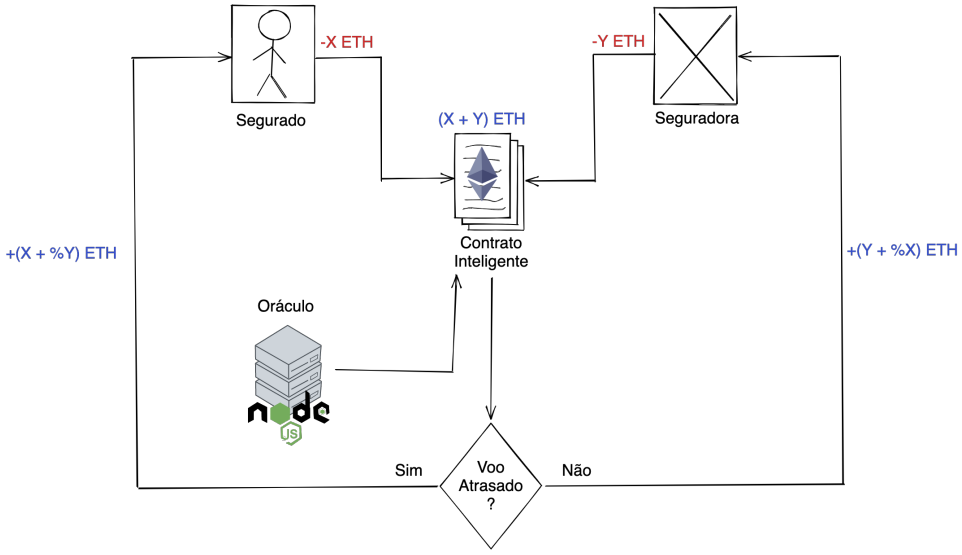
Para que o seguro seja estabelecido, o segurado deverá passar algumas informações referentes ao voo. Então, com base nas métricas predefinidas, os valores alocados pelo segurado

[1] Inclusive para seguros não paramétricos.

[2] Trata-se de um item opcional às seguradoras que oferecem seguro viagem.

e seguradora serão armazenados no *Smart Contract*, que cumprirá o papel de intermediário entre as partes. A relação será considerada finalizada quando o oráculo informar ao *Smart Contract* se o voo atrasou (incluindo-se cancelamento) ou não, resultando na execução da recompensa para a parte beneficiada, conforme esboçado pela figura 12.

Figura 12 – Proposta de Solução da Prova de Conceito




Fonte: o Autor

Não é objetivo deste trabalho demonstrar rigorosamente a sustentabilidade financeira do modelo, mas sim demonstrar, de maneira simplificada, a viabilidade técnica da implementação. Apesar disso, pode-se inferir que se trata de um modelo financeiramente sustentável, considerando-se que nem sempre acontecerão atrasos ou cancelamentos nos voos. Ainda, para garantir que, de fato, seja um modelo financeiramente viável, é possível construir modelos determinísticos de risco que maximizem o benefício para todas as partes envolvidas.

## 3.2 CONECTANDO OS CONCEITOS-CHAVES

Nesta altura do trabalho, espera-se o conhecimento de alguns conceitos fundamentais para o entendimento do que está acontecendo por trás da aplicação. Contudo, visando facilitar o entendimento, será feita uma breve definição dos conceitos essenciais utilizados para a construção da prova de conceito.

a) *Blockchain*: pode ser vista como um livro-razão, em que é armazenada uma lista completa dos registro de transações. É, em outras palavras, um sistema distribuído

*Peer-to-peer* que resolve dois problemas clássicos: gasto duplo e o Problema dos Generais Bizantinos. Espera-se que toda transação seja imutável e inviolável;

b) *Ethereum*: é uma rede *Blockchain* com uma linguagem de programação *Turing-complete* embutida, o que permite a construção de aplicações descentralizadas que vivem na rede *Ethereum*;

c) *Smart Contract*: o termo *Smart Contract* (Contrato Inteligente, em tradução livre), quando foi cunhado, visava definir um tipo de transação auto-executável. Inclusive, uma *vending machine* pode ser vista como uma ancestral do *Smart Contract*. Contudo, do ponto de vista da *Ethereum*, *Smart Contract* é um programa com funções e estados, rodando na sua *Blockchain*. Portanto, espera-se a sua execução sem a interferência de intermediários. Há diversos ganhos, como transação auto-executável, transparência, invulnerabilidade[3] etc;

d) Oráculo (*Oracle*): visando manter a natureza descentralizada possibilitada pela tecnologia *Blockchain*, Oráculos podem ser enxergados como fontes confiáveis de informações externas. Ou seja, é uma maneira de consultar o mundo externo ao *Blockchain* sem quebrar a premissa de descentralização[4];

e) *API*: é uma interface de programação de aplicações que contém o conjunto de especificações de possíveis interações entre aplicações;

f) *web3*: representa um novo paradigma da *web*, na qual o usuário lê, escreve e transaciona por meio de *tokens*.

## 3.3 PROVA DE CONCEITO

Feita a breve recapitulação, a seguir tem-se a discussão a respeito da modelagem e implementação da prova de conceito.

### 3.3.1 Definição da Arquitetura

A arquitetura desenhada para a prova de conceito proposta neste trabalho pode ser observada na figura 13.

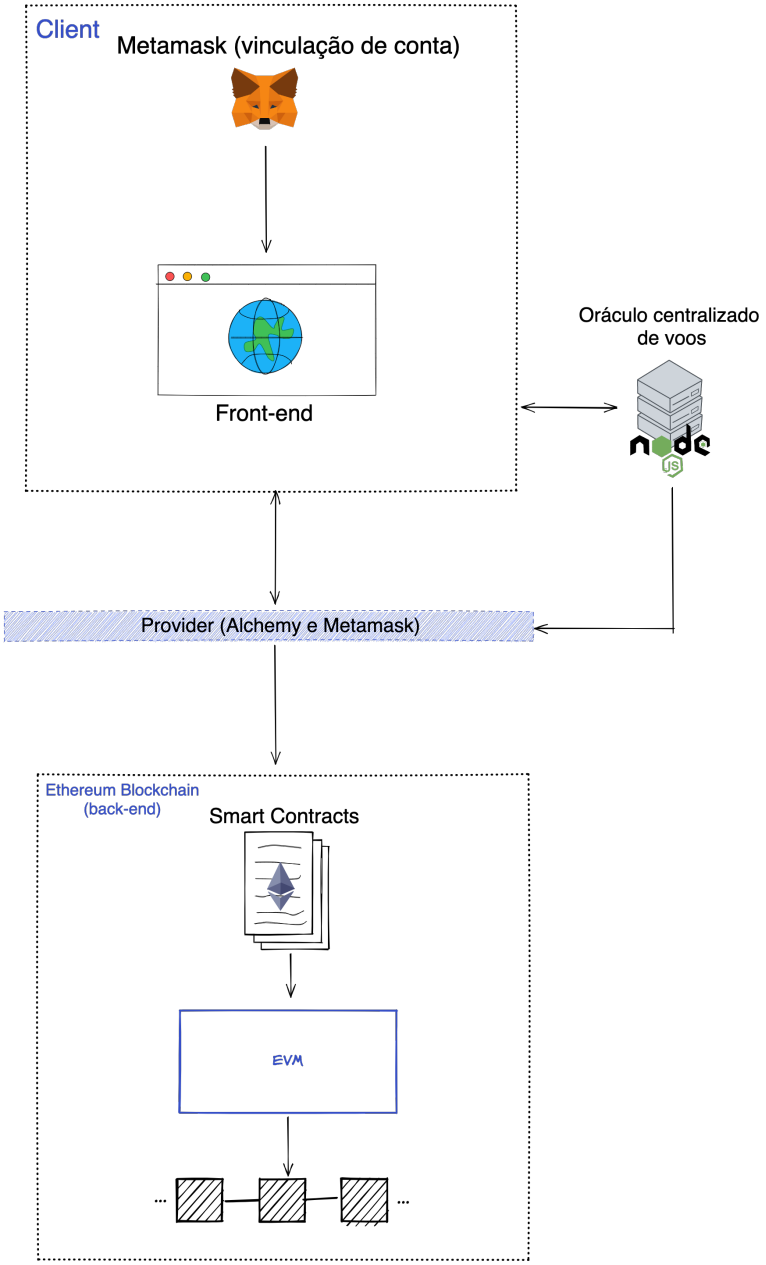
Por facilidade de implementação, o oráculo será mimetizado por meio de uma *API* de autoria própria. Portanto, trata-se de um oráculo centralizado. A *API* desenvolvida terá o papel

[3]Dependerá da implementação, dado que pode ser codificado de maneira a abrir espaços para invasores
[4]Dependerá da estratégia adotada

de informar o *Smart Contract* se o voo foi atrasado ou cancelado, além de alimentar a interface gráfica da aplicação (*front-end*) com detalhes do voo.

Idealmente, com a finalidade de garantir consenso da informação obtida, o *Smart Contract* receberia informação de vários oráculos diferentes e se apoiaria de algum projeto que viabiliza a manutenção da descentralização como, por exemplo, o *Chainlink*.

Figura 13 – Arquitetura da Prova de Conceito




Fonte: o Autor

Ainda com base na figura 13, é possível notar que não existe uma base de dados centralizada que armazena os estados da aplicação, nem um servidor *web* centralizado na qual a lógica da aplicação reside. Esta é uma das maiores diferenças entre *web2* e *web3*.

Pode-se dizer, considerando a adoção da rede *Ethereum* na arquitetura apresentada, que a aplicação está contida em uma máquina de estados descentralizada que é mantida por diversos nós anônimos na *Internet*. Ou seja, a aplicação não é mantida individualmente, mas coletivamente na rede.

O *Smart Contract* destacado na arquitetura, como já definido anteriormente neste trabalho, é um programa que roda na rede *Blockchain Ethereum*, definindo a lógica por trás das mudanças de estado que ocorrem na aplicação.

Os *Smart Contracts* são escritos em uma linguagem alto-nível. Então, para executar a lógica definida neles e realizar o processamento dos estados que ocorrem na rede *Blockchain*, existe a *Ethereum Virtual Machine* (*EVM*). Como ela não entende linguagem alto-nível, o código é compilado para *bytecode*.

Note a existência de mais alguns elementos na arquitetura. O *front-end* é bem parecido como implementado na *web2*, mas aqui ele também tem a responsabilidade de se comunicar com o *Smart Contract*. Comunicar-se com um *Smart Contract*, por sua vez, não é uma tarefa computacionalmente trivial. Portanto, para facilitar toda a integração existem os denominados *providers*.

Como já citado, uma aplicação *Blockchain* é mantida por nós na rede. Configurar e manter uma infraestrutura para rodar a aplicação é algo complexo. Então, para lidar com a parte de infraestrutura, a arquitetura adota um serviço terceirizado prestado pela *Alchemy*.

Todo *provider* implementa uma especificação *JSON*-RPC[5] que possibilita o front-end da aplicação se comunicar com o *Blockchain*. Uma vez que o *provider* esteja conectado à rede, é possível ler os estados armazenados nela. Agora, caso o objetivo seja escrever um novo estado, tem-se a necessidade de assinar a transação com uma chave privada, o que é possibilitado pelo *MetaMask*.

O MetaMask, nesta arquitetura, cumpre dois papeis: *provider* e assinante de transação. Ele cumpre o papel de *provider* porque ele tem uma conexão com os nós provisionados pela *Alchemy* e assinante porque é uma carteira cripto que possui a chave privada para assinar as transações na rede.

[5] É um protocolo de chamada de procedimento remoto (RPC) leve e sem estado

O último elemento a ser destacado nesta arquitetura é o oráculo. Apesar do nome diferente, trata-se simplesmente de uma aplicação que cumpre o papel de informar se um voo atrasou ou não. Ao receber a informação, há um tratamento no *front-end* e, por meio do *MetaMask*, é realizada a mudança de estado no *Blockchain*, resultando na execução das regras previstas no *Smart Contract*.

### 3.3.2 Desenvolvimento

Do ponto de vista de desenvolvimento, a prova de conceito pode ser divida em 3 partes: *Smart Contract*, *front-end* e *API* mimetizadora de oráculo. No Apêndice A, tem-se os principais códigos-fonte desenvolvidos e referências aos repositórios que contêm o projeto na íntegra. Aos interessados em saber as tecnologias utilizadas em detalhes, a visita aos repositórios é fortemente recomendada.

#### *3.3.2.1 Implantação do Smart Contract na Rede Ethereum*

A rede *Ethereum* possui o *Ether* como moeda nativa. Diferente do *Bitcoin*, o *Ether* tem um propósito maior do que ser apenas uma criptomoeda. Dada a possibilidade de construir aplicações que vivem na rede *Ethereum*, o *Ether* serve como uma recompensa para os mineradores validarem as transações que acontecem na rede. Ou seja, o *Ether* é uma forma de combustível para existência e manutenção da rede *Ethereum* (ETHEREUM, 2022).

A aquisição de *Ether* demanda o gasto verdadeiro de dinheiro. Então, visando contornar tal situação, existem as redes de teste que mimetizam o comportamento da rede verdadeira. Inclusive, algumas implementam o mesmo algoritmo de consenso (*Proof of Work*), que já foi discutido neste trabalho.

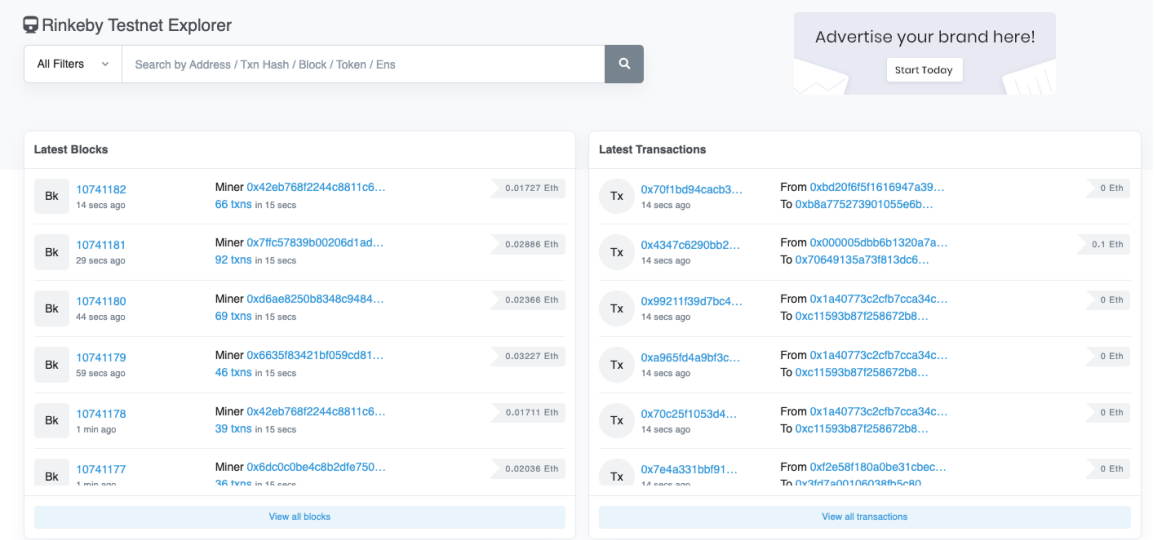
A prova de conceito está rodando na rede de teste *Rinkeby*. À semelhança da rede *Ethereum* principal, ela tem um livro-razão que registra todas as transações ocorridas, conforme pode ser observado na figura 14.

Como supramencionado, o *Ether* é usado para validar as transações na rede Ethereum. Nas redes de testes, é utilizado *Ether* falso gerado pelos denominados *faucets*. É necesseário possuir uma carteira cripto para armazenar *ether*. Alem disso, a carteira cripto é um pré-requisito para utilizar aplicações *web3*.

Na figura 15, pode-se observar um *faucet* e uma carteira cripto provisionada pela *Metamask* com alguns *ethers* falsos. A criação de uma carteira cripto é gratuita e possibilita interação com outras redes *Blockchain*, além da rede *Ethereum* e redes de teste.

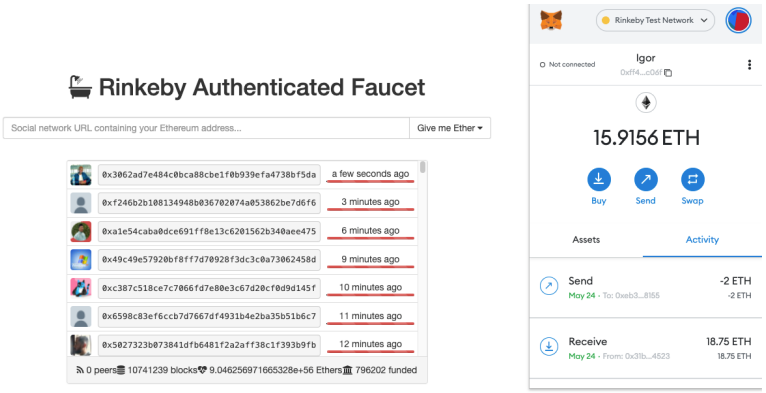
Uma das últimas etapas para conseguir fazer a implantação de um *Smart Contract* em uma rede Ethereum é possuir infraestrutura. Existem algumas plataformas que oferecem facilitação para desenvolver aplicações *Blockchain*. Então, como discutido na definição da arquitetura, a prova de conceito faz uso da *Alchemy*, uma plataforma de Infraestrutura como Código que abstrai muitas etapas da implantação e permite o desenvolvedor focar somente no código.

Figura 14 – Livro-Razão da Rede de Teste *Rinkeby*




Fonte: o Autor

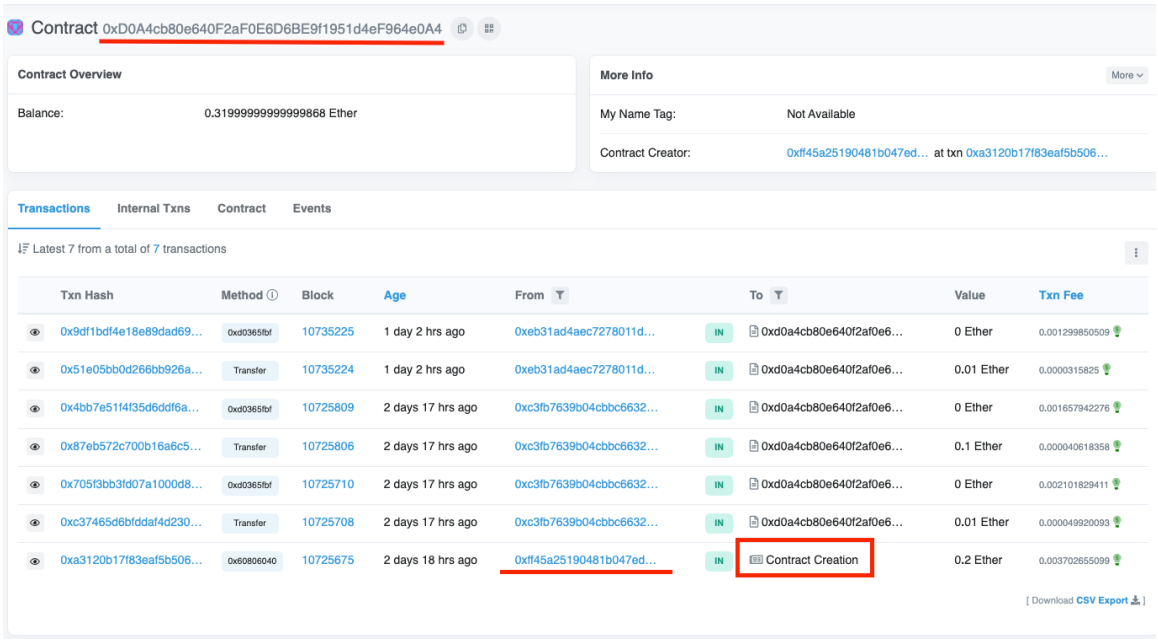
Figura 15 – Plataforma *Faucet* para Obtenção de *Ether* Falso




Fonte: o Autor

Reunindo-se todas as condições, com o suporte de um ambiente de desenvolvimento bem configurado, o *Smart Contract* pode, enfim, ser implantado na rede e ficar apto a receber interações. A figura 16 mostra o *Smart Contract* desenvolvido, cujo código-fonte está no apêndice A, no livro-razão da rede *Rinkeby*. A figura 16 ainda destaca o endereço do contrato[6], o criador do contrato e a informação de que ele foi criado.

Figura 16 – Registro do *Smart Contract* da Prova de Conceito na Rede Rinkeby




Fonte: o Autor

### *3.3.2.2 Construção da API Oráculo*

Para mimetizar o oráculo, foi construída uma *API* com informações de voos fictícios. Tal *api* tem a responsabilidade de informar se um determinado voo atrasou ou não, além de servir informações à interface gráfica da prova de conceito.

Trata-se de uma *API REST* simples, que foi escrita em JavaScript para o ambiente *NodeJS*. A aplicação está implantada por meio de serviço *Elastic BeanStalk* da *AWS* e se comunica com um banco de dados *PostgreSQL* implantado por meio do serviço RDS da *AWS*. O apêndice A inclui o código-fonte parcial e a referência ao repositório com o projeto na íntegra.

[6]Clique aqui para ser redirecionado ao livro-razão da rede *Rinkeby* com o contrato apresentado na figura 16.

O código-fonte a seguir mostra mostra a resposta de um dos *endpoints* da *API*. A resposta é em formato *json*. É possível notar que na resposta apresentada contém o número de voo, companhia aérea, data de partida, prêmio, recompensa e o *status* do voo (atrasado ou não).

A figura 17 os verbos *HTTP* implementados, destacando-se o *POST* com o corpo da requisição. No fim das contas, a *API* implementada é um CRUD[7] que permite criar voos, ler os voos registrados, atualizar o status do voo para cancelado e deletar um determinado voo.

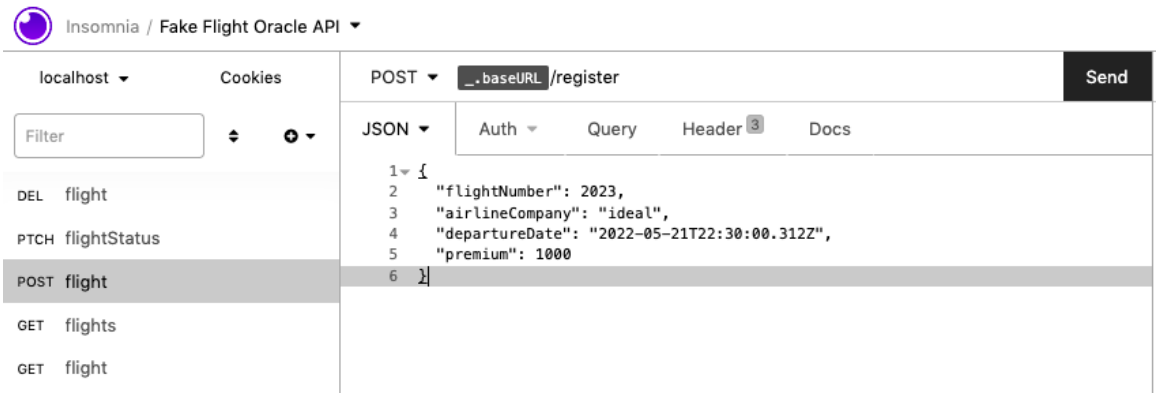

```
[
  {
    "flightnumber": 21,
    "airlinecompany": "ada",
    "departuredate": "2022-06-10T22:30:00.312Z",
    "premium": 100,
    "payout": 110,
    "isdelayedorcanceled": true
  },
  {
    "flightnumber": 43,
    "airlinecompany": "ada",
    "departuredate": "2022-06-10T22:30:00.312Z",
    "premium": 1000,
    "payout": 1100,
    "isdelayedorcanceled": false
  },
  {
    "flightnumber": 42,
    "airlinecompany": "ada",
    "departuredate": "2022-06-10T22:30:00.312Z",
    "premium": 1000,
    "payout": 1100,
    "isdelayedorcanceled": true
  },
  {
    "flightnumber": 2022,
    "airlinecompany": "carlos",
    "departuredate": "2022-06-10T22:30:00.312Z",
    "premium": 1000,
```


[7] Acrônimo para *Create*, *Read*, *Update* e *Delete*.

```
    "payout": 1100,
    "isdelayedorcanceled": true
  },
  {
    "flightnumber": 95,
    "airlinecompany": "gol",
    "departuredate": "2022-06-11T22:30:00.312Z",
    "premium": 100000,
    "payout": 110000,
    "isdelayedorcanceled": true
  },
  {
    "flightnumber": 33,
    "airlinecompany": "ideal",
    "departuredate": "2022-05-21T22:30:00.312Z",
    "premium": 1000,
    "payout": 1100,
    "isdelayedorcanceled": true
  }
]
```

Código-fonte 3.1 – Exemplo de Resposta da *API* Implantada na *AWS* para a Rota *Flights*

Figura 17 – Requisições Possíveis da *API*




Fonte: o Autor

#### *3.3.2.3 Construção da Inteface Gráfica*

Visando permitir a interação de usuários com a aplicação resultante da prova de conceito, foi construída uma interface gráfica bem simples e objetiva. Ela foi construída com o uso da biblioteca *ReactJS* do JavaScript, HTML e CSS. Tecnologias clássicas para construção do *front-end* de uma aplicação. Caso seja do interesse, é possível acessar ao repositório referenciado no apêndice A para visualizar o código-fonte.

A figura 18 mostra a tela que o usuário se deparará no primeiro acesso, em que ele ainda não conectou sua carteira cripto para poder interagir com a aplicação. As variações da tela poderão ser vistas no capítulo seguinte.

Figura 18 – Tela Inicial da Aplicação Resultante da Prova de Conceito


Fonte: o Autor

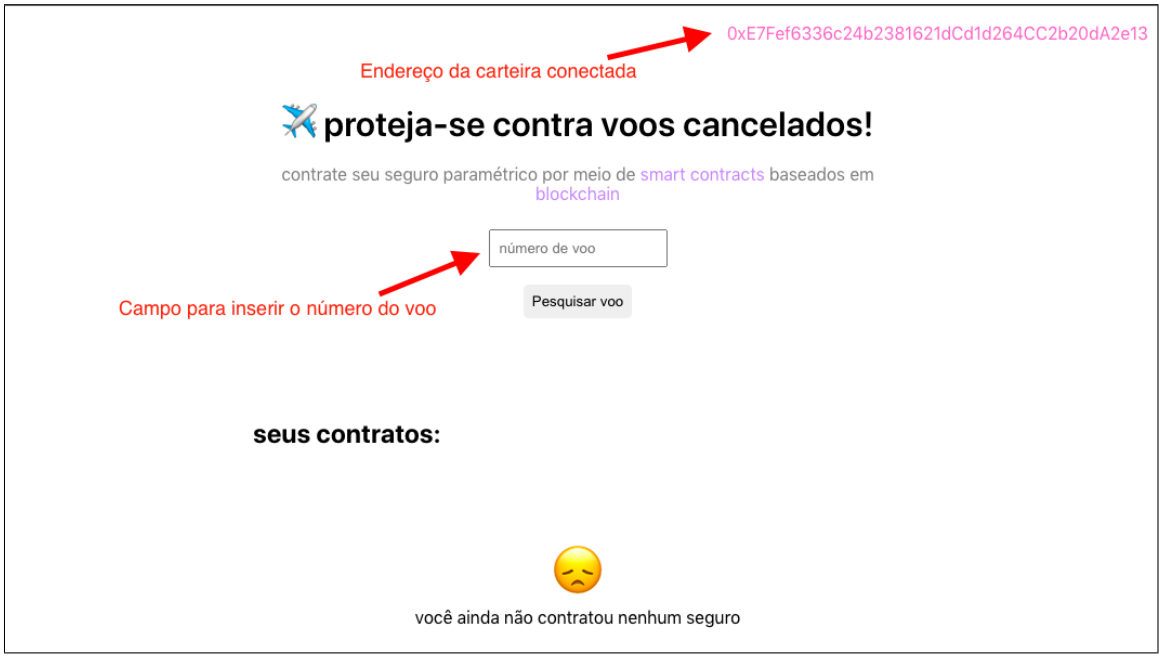
# 4 RESULTADOS

Com a delimitação de contexto definida e desenvolvimento da aplicação, foram obtidos resultados que corroboram a viabilidade técnica do uso de *Smart Contract* para contratação de seguros paramétricos descentralizados. A seguir, será mostrado o fluxo prático de contratação de um seguro para voos atrasados.

## 4.1 CONTRATAÇÃO DO SEGURO PARAMÉTRICO

Ao clicar no botão de se conectar, que pode ser observado na figura 18, o usuário terá a possibilidade de pesquisar voos inserindo o número dele e visualizar os seus contratos existentes, caso possua algum. A figura 19 ilustra a nova tela apresentada ao usuário, quando uma carteira é vinculada, destacando-se o endereço da carteira conectada à aplicação e o campo em que é inserido o número do voo.

Figura 19 – Tela do Usuário com a Carteira Conectada

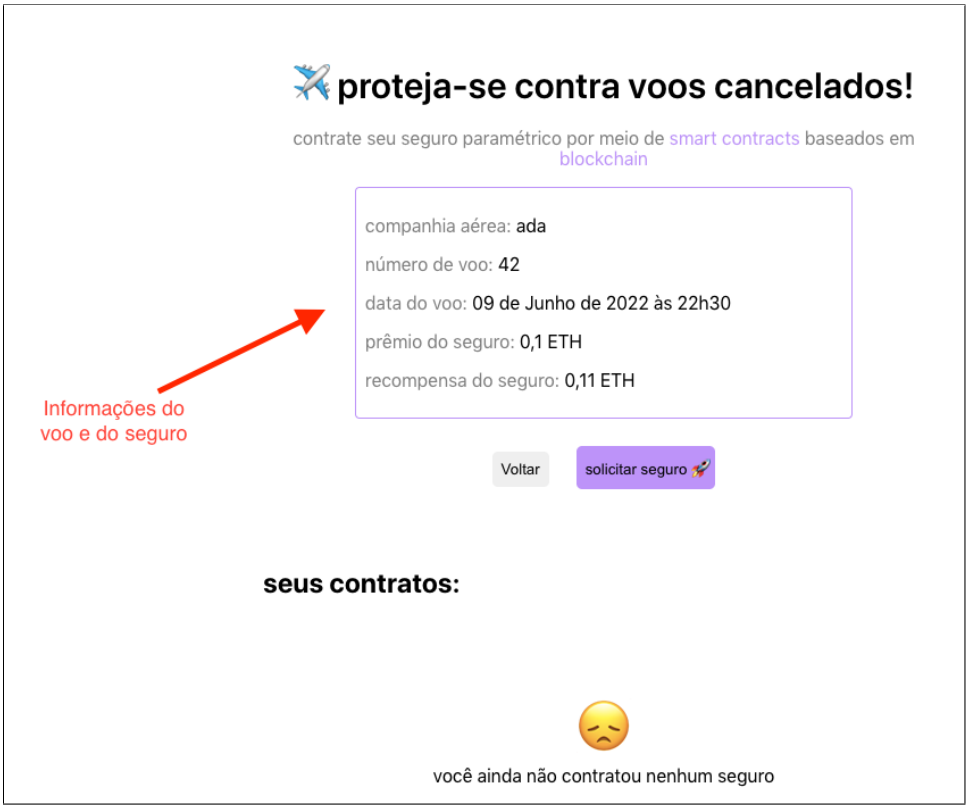



Fonte: o Autor

Ao inserir o número de um voo existente na *API*, será possível contratar o seguro para ele. Como já frisado no capítulo anterior, as informações de voos são fictícias e estão sendo provisionadas pela *API* oráculo. Como ilustra a figura 20, as informações associadas ao voo são mostradas e é apresentado o botão de solicitar seguro. Observe, ainda, que os valores de prêmio

e recompensa já são previamente definidos. Para a finalidade deste trabalho, a sustentabilidade econômica não está sendo levada em consideração, apenas a implementação técnica. Portanto, os valores apresentados para prêmio do seguro e recompensa têm apenas um papel figurativo. Além disso, é válido ressaltar que os valores estão sendo cotados em *ether* (ETH), a criptomoeda nativa da rede *Blockchain*.

Figura 20 – Tela com Informações do Voo e Seguro




Fonte: o Autor

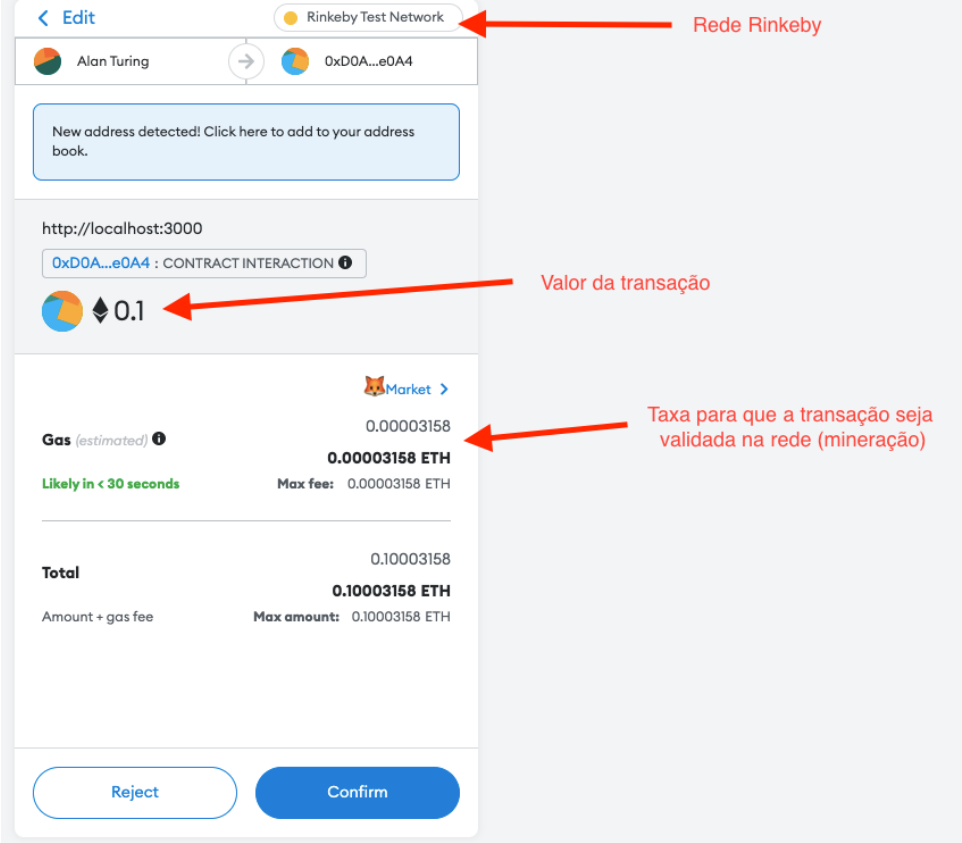
Então, ao clicar no botão de solicitar seguro, a *Metamask* entra em ação e permite que o usuário realize a transação e interaja com o *Smart Contract*. A estratégia do *Smart Contract* construído para esta prova de conceito é ser uma fábrica de contratos inteligentes. O contrato pai, que é mantido pela seguradora, consegue gerar vários contratos filhos. A lógica de tal implementação pode ser observada no código-fonte presente no apêndice A.

O usuário terá duas interações com a *Metamask* para que o *Smart Contract* atribuído ao voo seja criado. A primeira interação é para pagar o prêmio do seguro à seguradora, ou seja, o valor do prêmio é alocado no contrato pai. A segunda transação é responsável por criar o contrato inteligente de seguro associado ao voo (contrato filho) e pertencente ao segurado. Quando um

contrato filho é criado, o valor da recompensa do seguro é retirado do saldo do contrato pai e alocado no saldo do contrato gerado para o voo.

A figura 21 mostra a interação com a *Metamask* e destaca algumas informações importantes a respeito da transação, como: rede em que ela está ocorrendo, valor da transação (prêmio do seguro) e a taxa de combustível (*gas fee*), que foi melhor explicada na subseção 2.1.6.4. Frisando novamente o não foco na sustentabilidade financeira do modelo, a taxa de combustível é uma questão que deve ser melhor tratada considerando a aplicação em uma situação real, visto que o valor do prêmio e recompensa do seguro não leva tal taxa em consideração.

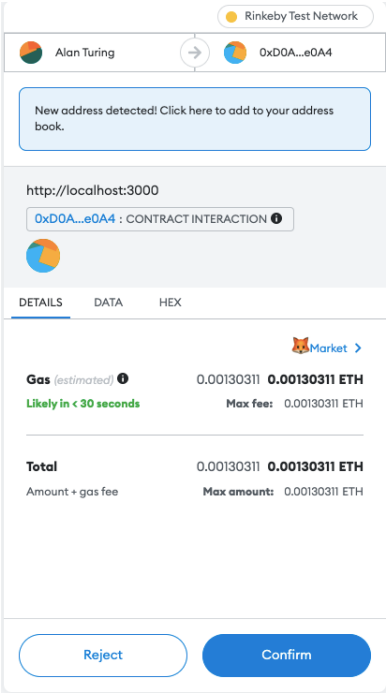
Figura 21 – Realização da Transação para Depositar o Valor do Prêmio




Fonte: o Autor

A figura 22 mostra a segunda interação, cujo o contrato atribuído ao voo é criado. É neste momento que o contrato pai envia parte do seu saldo (o valor da recompensa do seguro) para o contrato filho que está sendo gerado.

Figura 22 – Realização da Transação para Criar o Contrato Atribuído ao Voo




Fonte: o Autor

## 4.2 REGISTRO DO SEGURO PARAMÉTRICO NA REDE RINKEBY

Cada interação com a Metamask representa uma transação na rede *Rinkeby* e, portanto, um registro no livro-razão. A figura 23 mostra o contrato pai na rede *Rinkeby* e as transações que ocorreram nele. Ainda, na figura 23, destacam-se as duas últimas transações referenciadas nas figuras 21 e 22 e o saldo do contrato que houve alteração se comparado à figura 16.

## 4.3 SOLICITAÇÃO DE RECOMPENSA

Uma recompensa de seguro só é solicitável para voos que já aconteceram, visto que o objetivo é ser beneficiado em caso de atraso ou cancelamento. Para realizar tal demonstração, será utilizada uma conta que possui um contrato passível de análise.

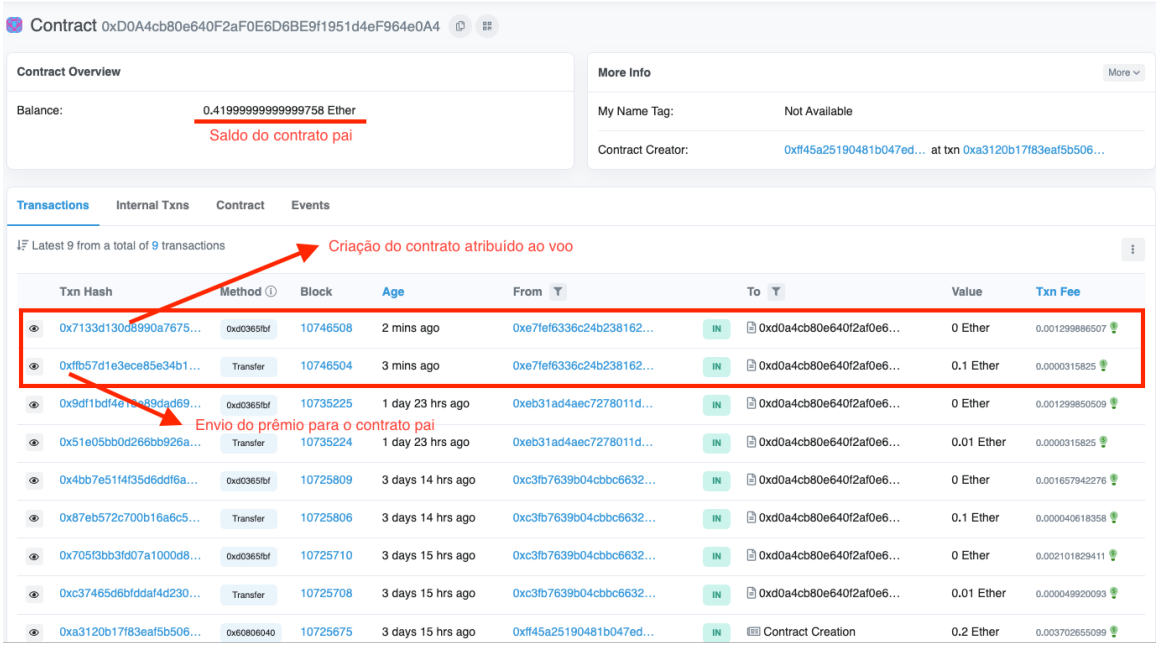
A interface gráfica indica quando um contrato já pode ser analisado, conforme mostra a figura 24, que também destaca a carteira com o saldo atual do segurado para fins de comparações a seguir.

Ao clicar em solicitar análise, o contrato atribuído ao voo recebe a informação do status do voo. Tal informação é fornecida pela *API* oráculo. Então, em caso de atraso ou cancelamento,

o valor da recompensa do seguro é enviado à carteira do segurado. Caso contrário, o valor é enviado à carteira da seguradora.

Simulando um cenário na qual o segurado é beneficiado, isto é, houve atraso ou cancelamento do voo, a figura 25 mostra o novo saldo da carteira, com o acréscimo da recompensa do seguro[1], e a atualização do estado do contrato na interface gráfica com a informação de que ele já foi analisado e, portanto, encerrado.

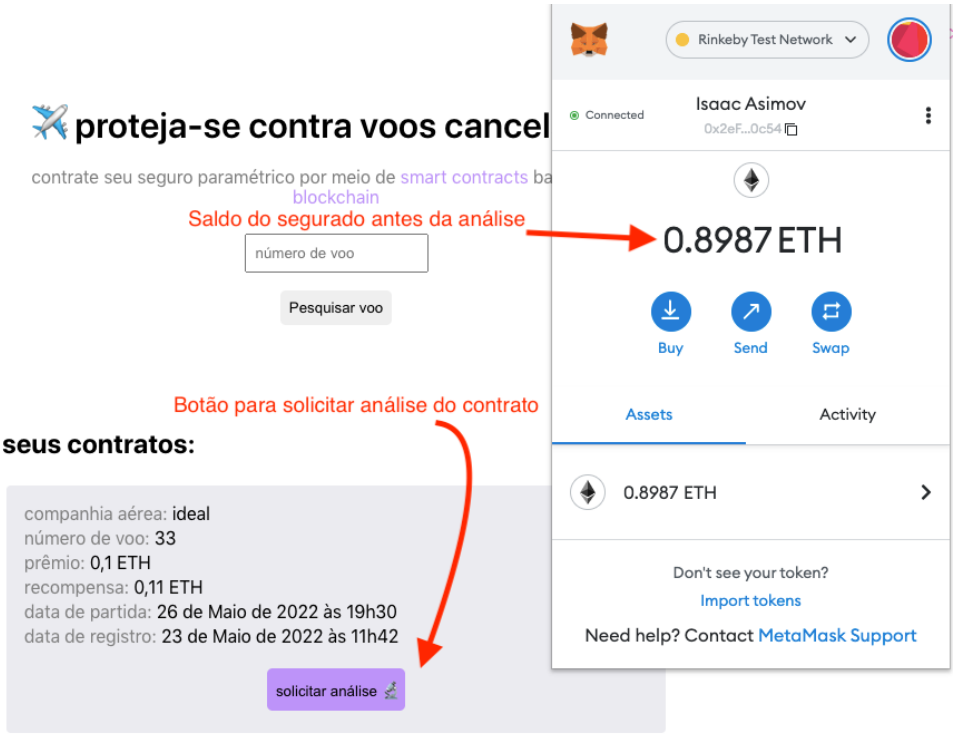
Figura 23 – Transações Registradas no Livro-razão da Rede Rinkeby




Fonte: o Autor

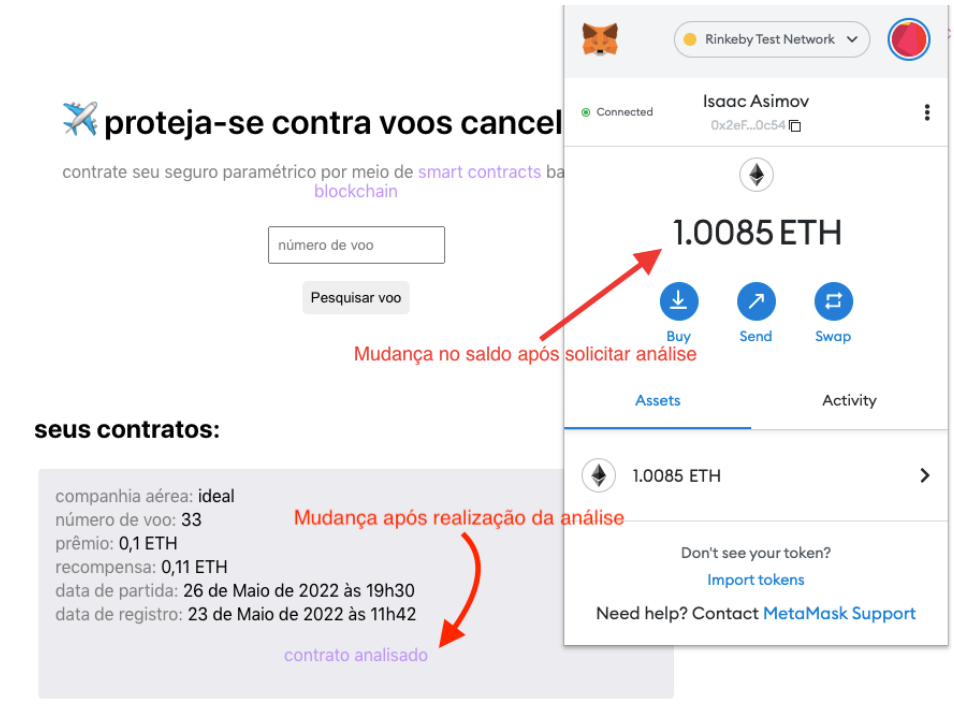
[1]Há um pequeno desconto devido à taxa *gas fee* cobrada pela rede para realizar a transação que informa ao contrato o *status* do voo.

Figura 24 – Contrato Disponível para Solicitar Seguro




Fonte: o Autor

Figura 25 – Contrato Analisado após Solicitação




Fonte: o Autor

# 5 CONCLUSÃO

As figuras apresentadas no capítulo anterior ilustram o processo de contratação e solicitação de recompensa de seguro na prova de conceito. Nota-se que é um processo extremamente simples, mas com um resultado bem significativo, visto que que as interações entre contas e contratos sem um intermediário são comprovadas. É evidente que existem muitas abstrações e simplificações, principalmente no tocante à sustentabilidade financeira do modelo e ao consumo de informação externa à rede *Blockchain* por meio de oráculos, mas a prova de conceito consegue cumprir o objetivo proposto: demonstrar a viabilidade técnica do uso de *Smart Contracts* para seguros paramétricos descentralizados.

Pensando no cenário de produção para um seguro paramétrico descentralizado contra atrasos ou cancelamentos de voos, que foi o produto considerado pela prova de conceito, muitas camadas precisam ser acrescidas. As mais relevantes, em conjunto com a refatoração do contrato para que o mesmo tenha mais complexidade e segurança, é o uso de alguma rede de oráculo *Blockchain* descentralizada, visando aumentar a confiabilidade e garantia de neutralidade das relações entre segurado e seguradora e a adoção de *API*s com informações reais de voos. Para a questão do oráculo descentralizado, existem no mercado algumas soluções, sendo *Chainlink* e *Provable*, no momento de escrita deste trabalho, as mais populares.

Um outro aspecto que pode ser melhor desenvolvido é quanto à viabilidade econômica. A prova de conceito leva somente a viabilidade técnica em consideração. Então, para chegar à conclusão de que se trata de um modelo de negócio factível em sua totalidade, necessita-se demonstrar a viabilidade financeira.

## 5.1 TRABALHOS FUTUROS

A partir deste trabalho, nasce a possibilidade para muitos trabalhos futuros. Alguns diretamente relacionados ao que foi discutido aqui e outros que podem usar alguns assuntos abordados como fagulha inicial para uma linha de pesquisa bem diferente. Portanto, pode-se dividir em trabalhos futuros que se relacionam diretamente com esta linha de pesquisa e trabalhos futuros que se derivam desta linha de pesquisa.

Começando pelos que se relacionam diretamente:

a) Construção de modelos determinísticos de risco para assegurar a viabilidade financeira dos seguros paramétricos descentralizados;

b) Construção de uma prova de conceito com o uso de oráculos descentralizados;
c) Análise sociológica e econômica da existência de um modelo de seguro descentralizado;
d) Construção de uma rede *Blockchain* privada;
e) Exploração de falhas de seguranças nos *Smart Contracts*;
f) Uso de *Smart Contracts* para seguros paramétricos agrícolas.

Trabalhos indiretamente relacionados:

a) Análise aprofundada dos algoritmos de consenso;
b) Aplicação de *Blockchain* e *Smart Contracts* em outras indústrias;
c) Desenvolvimentos de oráculos *Blockchain* baseados em *hardware*
d) Estratégias para a construção de oráculos *Blockchain*;

São muitas as possibilidades. Acima, têm-se algumas julgadas extremamente interessantes pelo autor.

## 5.2 CONSIDERAÇÕES FINAIS

O presente trabalho começou com a proposição de um projeto que dialogasse com os recentes avanços promovidos pelo *Open Insurance*, um análogo do *Open Banking* para a indústria de seguros. A ideia inicial era a construção de um comparador de seguros baseado em sistemas de recomendação. Tratava-se de uma proposta motivada pela vontade de propor produtos e serviços que coloquem o cliente no centro.

À medida que os estudos específicos para o desenvolvimento do comparador iam evoluindo, notava-se uma grande dificuldade, dada a dependência da evolução das fases do *Open Insurance* e toda a regulação imposta pela SUSEP.

Então, ao estudar o conceito de *Smart Contract*, a direção do trabalho mudou e uma nova ideia surgiu, resultando-se neste trabalho: uma prova de conceito, que apesar de bem diferente de um comparador, mantém o sentimento inicial de querer colocar o consumidor no centro, permitindo-o contratar um serviço de seguro transparente, objetivo e sem centralização de poder.

Ao abordar a ideia de *Smart Contracts* baseados em *Blockchain*, um assunto de muito potencial tecnológico-científico, espera-se abrir espaço para o nascimento de discussões ricas e soluções que tornem o mundo um lugar mais equitativo.

**APÊNDICE A –** CÓDIGOS-FONTE DA PROVA DE CONCEITO

A concepção do projeto como um todo pode ser vista nos seguintes repositórios do *Github* (clique para ser redirecionado):

a) Aplicação (*Smart Contract* e *front-end*)

b) *API* (oráculo mimetizado)

A seguir, têm-se os códigos-fonte julgados mais relevantes do projeto.

```
pragma solidity ^0.8.0;

contract Insurance {
    address public insurer;
    address public insured;
    string public airlineCompany;
    uint256 public flightNumber;
    uint256 public premium;
    uint256 public payout;
    bool public isFlightDelayed;
    uint256 public departureDate;
    uint256 public timestamp;
    bool isOracleAlreadyCalled = false;

    constructor(
        address _insurer,
        address _insured,
        string memory _airlineCompany,
        uint256 _flightNumber,
        uint256 _premium,
        uint256 _payout,
        uint256 _departureDate
    ) public {
        insurer = _insurer;
        insured = _insured;
        airlineCompany = _airlineCompany;
        flightNumber = _flightNumber;
        premium = _premium;
        payout = _payout;
        departureDate = _departureDate;
        payout = _payout;
        isFlightDelayed = false;
        timestamp = block.timestamp;
```

```
34      }
35
36      function transferPayout(address payable recipient) public payable {
37          require(msg.sender == insured);
38
39          recipient.transfer(payout);
40      }
41
42      function callOracle(bool _oracleInformation) public {
43          require(msg.sender == insured);
44          require(!isOracleAlreadyCalled);
45
46          isFlightDelayed = _oracleInformation;
47
48          if (isFlightDelayed) {
49              transferPayout(payable(insured));
50          } else {
51              transferPayout(payable(insurer));
52          }
53
54          isOracleAlreadyCalled = true;
55      }
56
57      function getBalance() public view returns (uint256) {
58          return address(this).balance;
59      }
60
61      receive() external payable {}
62  }
63
64  contract InsuranceFactory {
65      address provider = msg.sender;
66      address providerContractAddress = address(this);
67      bool isContractActive = true;
68
69      struct InsuranceInformations {
70          address contractAddress;
71          address insured;
72          string airlineCompany;
```

```
73          uint256 flightNumber;
74          uint256 premium;
75          uint256 payout;
76          uint256 departureDate;
77          uint256 timestamp;
78      }
79
80      InsuranceInformations[] insurances;
81
82      constructor() payable {}
83
84      event NewInsurance(
85          address indexed insured,
86          string airlineCompany,
87          uint256 flightNumber,
88          uint256 premium,
89          uint256 payout,
90          uint256 departureDate,
91          uint256 timestamp
92      );
93
94      function createInsurance(
95          string memory _airlineCompany,
96          uint256 _flightNumber,
97          uint256 _premium,
98          uint256 _payout,
99          uint256 _departureDate
100     ) public payable {
101         require(isContractActive);
102         require(hasBalanceToCoverPayout(_payout));
103
104         Insurance insurance = new Insurance(
105             provider,
106             msg.sender,
107             _airlineCompany,
108             _flightNumber,
109             _premium,
110             _payout,
111             _departureDate
```

```
112         );
113
114         insurances.push(
115             InsuranceInformations(
116                 address(insurance),
117                 msg.sender,
118                 _airlineCompany,
119                 _flightNumber,
120                 _premium,
121                 _payout,
122                 _departureDate,
123                 block.timestamp
124             )
125         );
126
127         emit NewInsurance(
128             msg.sender,
129             _airlineCompany,
130             _flightNumber,
131             _premium,
132             _payout,
133             _departureDate,
134             block.timestamp
135         );
136
137         (bool success, ) = (address(insurance)).call{value: _payout}("")
                ;
138         require(success, "Failed to withdraw money from contract.");
139     }
140
141     function getAllInsurances()
142         public
143         view
144         returns (InsuranceInformations[] memory)
145     {
146         return insurances;
147     }
148
149     function getProvider() public view returns (address) {
```

```
150         return provider;
151     }
152 
153     function getResidualEth() public payable {
154         require(msg.sender == provider);
155 
156         payable(provider).transfer(address(this).balance);
157     }
158 
159     function getBalance() public view returns (uint256) {
160         return address(this).balance;
161     }
162 
163     function endContract() public {
164         require(msg.sender == provider);
165         isContractActive = false;
166         getResidualEth();
167     }
168 
169     function hasBalanceToCoverPayout(uint256 _payout)
170         public
171         view
172         returns (bool)
173     {
174         if (_payout >= address(providerContractAddress).balance) {
175             return false;
176         }
177 
178         return true;
179     }
180 
181     receive() external payable {}
182 }
```

Código-fonte A.1 – *Smart Contract* da Prova de Conceito em Solidity

```
1 const express = require('express')
2 const auth = require('../middlewares/auth')
3 const { PrismaClient } = require('@prisma/client')
4 
```

```
5  const prisma = new PrismaClient()
6
7  const flightsRouter = express.Router()
8
9  flightsRouter.post('/register', auth, async (req, res, next) => {
10   try {
11     const { flightNumber, airlineCompany, email, departureDate, premium
          } =
12       req.body
13
14     const flight = await prisma.flight.create({
15       data: {
16         flightnumber: flightNumber,
17         airlinecompany: airlineCompany,
18         email: email,
19         departuredate: departureDate,
20         premium: premium,
21         payout: premium + premium * 0.1,
22         isdelayedorcanceled: false,
23       },
24     })
25
26     return res.status(201).json(flight)
27   } catch (error) {
28     res.status(500).send({
29       message: 'It was not possible to register a flight!',
30       error,
31     })
32   }
33 })
34
35 flightsRouter.get('/', async (req, res, next) => {
36   const flights = await prisma.flight.findMany()
37
38   return res.json(flights)
39 })
40
41 flightsRouter.get('/:flightNumber', async (req, res, next) => {
42   const { flightNumber } = req.params
```

```
43
44    try {
45      const flight = await prisma.flight.findUnique({
46        where: { flightnumber: Number(flightNumber) },
47      })
48
49      if (!flight) throw new Error('Not found!')
50
51      return res.json(flight)
52    } catch (error) {
53      return res
54        .status(404)
55        .json({ message: "this flight number doesn't exist", error })
56    }
57  })
58
59  flightsRouter.patch('/:flightNumber/delayed', auth, async (req, res,
        next) => {
60    const { flightNumber } = req.params
61
62    try {
63      const flight = await prisma.flight.update({
64        where: { flightnumber: Number(flightNumber) },
65        data: {
66          isdelayedorcanceled: true,
67        },
68      })
69
70      return res.status(204).send()
71    } catch (error) {
72      return res
73        .status(400)
74        .json({ message: "this flight number doesn't exist", error })
75    }
76  })
77
78  flightsRouter.delete('/:flightNumber', async (req, res, next) => {
79    const { flightNumber } = req.params
80
```

```
81    try {
82      const flight = await prisma.flight.delete({
83        where: { flightnumber: Number(flightNumber) },
84      })
85
86      return res.status(204).send()
87    } catch (error) {
88      return res
89        .status(400)
90        .json({ message: "this flight number doesn't exist", error })
91    }
92  })
93
94  module.exports = flightsRouter
```

Código-fonte A.2 – Código Parcial da API Mimetizadora de Oráculo em *JavaScript*

# REFERÊNCIAS

BACK, Adam et al. Hashcash-a denial of service counter-measure, 2002.

BAMAKAN, Seyed Mojtaba Hosseini; MOTAVALI, Amirhossein; BABAEI BONDARTI, Alireza. A survey of blockchain consensus algorithms performance evaluation criteria. **Expert Systems with Applications**, v. 154, p. 113385, 2020. ISSN 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113385. Disponível em: https://www.sciencedirect.com/science/article/pii/S0957417420302098.

BENIICHE, Abdeljalil. A study of blockchain oracles. **arXiv preprint arXiv:2004.07140**, 2020.

BHARGAVA, Aditya. **Grokking Algorithms: An illustrated guide for programmers and other curious people**. [S.l.]: Simon e Schuster, 2016.

BODÓ, Balázs; GIANNOPOULOU, Alexandra. The logics of technology decentralization–the case of distributed ledger technologies. **Blockchain and web**, v. 3, 2019.

BUTERIN, Vitalik. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform, 2013. Disponível em: https://ethereum.org/en/whitepaper/.

CHESBROUGH, Henry William. **Open innovation: The new imperative for creating and profiting from technology**. [S.l.]: Harvard Business Press, 2003.

CHUEN, David. **Handbook of Digital Currency: Bitcoin, Innovation, Financial Instruments, and Big Data**. [S.l.: s.n.], jan. 2015. P. 1–588.

CLACK, Christopher D; BAKSHI, Vikram A; BRAINE, Lee. Smart contract templates: foundations, design landscape and research directions. **arXiv preprint arXiv:1608.00771**, 2016.

COHN, Alan; WEST, Travis; PARKER, Chelsea. Smart after all: blockchain, smart contracts, parametric insurance, and smart energy grids. **Geo. L. Tech. Rev.**, HeinOnline, v. 1, p. 273–303, 2016.

ETHEREUM. **Documentação Ethereum**. [S.l.: s.n.]. https://ethereum.org/en/developers/docs/. Acesso em: 2022.

EXCELÊNCIA EM GESTÃO, Congresso Nacional de. **Panorama Estatístico | CNseg - O Portal do Seguro**. [S.l.: s.n.]. https://cnseg.org.br/analises-e-estatisticas/panorama-estatistico-8A8AA8A37A39FA6A017AC49227594A3A.html. Acesso em: 2022.

EXCELÊNCIA EM GESTÃO, Congresso Nacional de. **Panorama Estatístico | CNseg - O Portal do Seguro**. [S.l.: s.n.]. https://cnseg.org.br/noticias/cnseg-divulga-novo-ranking-das-empresas-do-setor-de-seguros.html. Acesso em: 2022.

GLOBO, O. **Sites identificam buscas de consumidor e aumentam preços de produtos em até 20% - Jornal O Globo**. [S.l.: s.n.].

https://oglobo.globo.com/economia/defesa-do-consumidor/sites-identificam-buscas-de-consumidor-aumentam-precos-de-produtos-em-ate-20-22484138. Acesso em: 2022.

HABER, Stuart; STORNETTA, W Scott. How to time-stamp a digital document. In: SPRINGER. CONFERENCE on the Theory and Application of Cryptography. [S.l.: s.n.], 1990. P. 437–455.

HUMBY, Clive. Data is the new oil. **Proc. ANA Sr. Marketer's Summit. Evanston, IL, USA**, 2006.

IBARRA, Hector; SECURITIES, Insurance Linked. Parametric insurance: general market trends and perspectives for the African insurance sector. **Insurance Linked Securities. Partnerre New Solutions Inc**, 2010.

INVESTE, Valor. **Preço de produtos em sites muda quando pesquisado em iPhone, Android e computador | Gastar Bem | Valor Investe**. [S.l.: s.n.]. https://valorinveste.globo.com/objetivo/gastar-bem/noticia/2019/10/07/preco-de-produtos-em-sites-muda-quando-pesquisado-em-iphone-android-e-computador.ghtml. Acesso em: 2022.

LIU, Zhuotao et al. Make Web3. 0 Connected. **IEEE Transactions on Dependable and Secure Computing**, IEEE, 2021.

LOUKIL, Faiza et al. CioSy: A Collaborative Blockchain-Based Insurance System. **Electronics**, v. 10, n. 11, 2021. ISSN 2079-9292. DOI: 10.3390/electronics10111343. Disponível em: https://www.mdpi.com/2079-9292/10/11/1343.

MEAD, N.R. et al. **Software Security Engineering: A Guide for Project Managers**. [S.l.]: Pearson Education, 2004. P. 115–116. (SEI Series in Software Engineering). ISBN 9780132702454. Disponível em: https://books.google.com.br/books?id=hl-zvFPQAfcC. Acesso em: 2022.

MINGXIAO, Du et al. A review on consensus algorithm of blockchain. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC). [S.l.: s.n.], 2017. P. 2567–2572. DOI: 10.1109/SMC.2017.8123011.

MUSSENBROCK, C; KARPISCHEK, S. Etherisc Whitepaper. 2018. **URL: https://www. etherisc. com/whitepaper (Visitado em 2022.)**

NAKAMOTO, Satoshi. **Bitcoin: A Peer-to-Peer Electronic Cash System**. [S.l.: s.n.], dez. 2008. Accessed: 2015-07-01. Disponível em: https://bitcoin.org/bitcoin.pdf.

POVO, Gazeta do. **Empresas usam cada vez mais big data para conhecer seus clientes**. [S.l.: s.n.]. https://www.gazetadopovo.com.br/economia/empresas-usam-cada-vez-mais-big-data-para-conhecer-seus-clientes-af63d1v4hdin7sku6q5xi1cbd/. Acesso em: 2022.

SANTANDER. **Conheça DeFi: o protocolo que veio para revolucionar o mundo das criptomoedas**. [S.l.: s.n.]. https://santandernegocioseempresas.com.br/conhecimento/gestao-financeira/defi/. Acesso em: 2022.

SCHERRER, John; SALAHSHOR, Abtin. **Smart Contracts, Insurtechs and the Future of Insurance**. eng. [S.l.: s.n.], 2020. Student Paper.

SZABO, Nick. Formalizing and securing relationships on public networks. **First monday**, 1997.

TEK, SAPO. **Concentração da Inteligência Artificial nas big tech traz riscos de dependência da economia e dos Estados - Internet - SAPO Tek**. [S.l.: s.n.]. https://tek.sapo.pt/noticias/internet/artigos/concentracao-da-inteligencia-artificial-nas-big-tech-traz-riscos-de-dependencia-da-economia-e-dos-estados. Acesso em: 2022.

VOSHMGIR, S. **Token Economy: How the Web3 reinvents the Internet**. [S.l.]: Shermin Voshmgir, 2020. ISBN 9783982103839. Disponível em: https://books.google.com.br/books?id=vWo-EAAAQBAJ.

VUJIČIĆ, Dejan; JAGODIĆ, Dijana; RANĐIĆ, Siniša. Blockchain technology, bitcoin, and Ethereum: A brief overview. In: IEEE. 2018 17th international symposium infoteh-jahorina (infoteh). [S.l.: s.n.], 2018. P. 1–6.

WOOD, Gavin et al. Ethereum: A secure decentralised generalised transaction ledger. **Ethereum project yellow paper**, v. 151, n. 2014, p. 1–32, 2014.

ZARRIN, Javad et al. Blockchain for decentralization of internet: prospects, trends, and challenges. **Cluster Computing**, Springer, v. 24, n. 4, p. 2841–2866, 2021.

ZHENG, Z et al. Blockchain Challenges and Opportunities: A Survey; Work Paper. **Inderscience Publishers: Geneva, Switzerland**, 2016.