INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO NORTE
CAMPUS NATAL CENTRAL
DIRETORIA ACADÊMICA DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO

YURI HENRIQUE SALES DA COSTA

ANÁLISE COMPARATIVA DA COMUNICAÇÃO DE EXCHANGES DE CRIPTOMOEDAS

NATAL
2019

YURI HENRIQUE SALES DA COSTA

ANÁLISE COMPARATIVA DA COMUNICAÇÃO DE EXCHANGES DE CRIPTOMOEDAS

Trabalho de conclusão de curso apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Diretoria Acadêmica de Gestão e Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Dra. Cláudia Maria Fernandes Araújo Ribeiro

NATAL
2019

YURI HENRIQUE SALES DA COSTA

ANÁLISE COMPARATIVA DA COMUNICAÇÃO DE EXCHANGES DE CRIPTOMOEDAS

Trabalho de conclusão de curso apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Diretoria Acadêmica de Gestão e Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho de Conclusão de Curso apresentado e aprovado em ___/___/____, pela seguinte Banca Examinadora:

BANCA EXAMINADORA

Cláudia Maria Fernandes Araújo Ribeiro – Presidente

Fellipe Araújo Aleixo – Examinador

Leonardo Ataide Minora – Examinador

NATAL
2019

Aos meus avós.

AGRADECIMENTOS

À minha orientadora Cláudia Maria Fernandes Araújo Ribeiro, que aceitou orientar esta pesquisa e deu todo o suporte necessário para que ela fosse concluída.

Ao meu companheiro de trabalho e coorientador Giovani Ângelo Silva da Nóbrega, que foi o principal contribuinte para a realização desta pesquisa.

Aos meus familiares, principalmente à minha avó Elcy Silva Sales, por todo o investimento na minha educação.

Nunca e em lugar algum do universo existe estabilidade e imobilidade. Mudança e transformação são características essenciais da vida. Cada estado de coisas é passageiro; cada época é uma época de transição. Na vida humana nunca há calma e repouso. A vida é um processo e não a permanência no status quo.

Ludwig von Mises

RESUMO

Com a popularização do *Bitcoin* e de outras criptomoedas, casas de câmbio *online* vêm surgindo para que os usuários possam comprar, guardar, transacionar e operar no mercado suas moedas digitais. Devido a esse aumento na oferta de *exchanges* e de moedas, é possível desenvolver estratégias de mercado para compra e venda em diferentes lugares entre diferentes criptomoedas; e uma forma de facilitar esse processo é por meio da automatização do mesmo. Tal automatização pode ser concebida através da construção de aplicações que consomem as *APIs* das casas de câmbio. Essas *APIs* ainda carecem de consistência de informações, qualidade, materiais de apoio e são repletas de limitações. Devido a essa imaturidade, desenvolvedores encontram dificuldade em trabalhar com essas *APIs*. Este trabalho tem como objetivo realizar uma análise comparativa entre seis corretoras, utilizando como parâmetro os padrões de desenvolvimento e as boas práticas de *design* de *APIs*, assim como a verificação de suas limitações, e documentar os resultados a fim de auxiliar outros desenvolvedores.

Palavras-chave: Modelagem de APIs. REST. Exchanges de Criptomoedas.

ABSTRACT

With the popularization of *Bitcoin* and others cryptocurrencies, online exchange offices have been emerging so that users can buy, store, transact and operate in the market their digital coins. Due to this increase in the supply of exchanges and currencies, it is possible to develop market strategies to buy and sell in different places between different cryptocurrencies; and a way to facilitate this process is by automating itself. Such automation can be conceived through the development of applications which consume the exchange offices *APIs*. These *APIs* still lack information consistency, quality, support materials and they are full of restrictions. Due to this immaturity, developers find it difficult to work with these *APIs*. This paper aims to perform a comparative analysis of the six main digital currencies exchange offices using as a parameter the development standards and the *API* design good practices, also checking their limitations and documenting the results in order to assist others developers.
Keywords: APIs Design. REST. Cryptocurrencies Exchanges.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O dinheiro pode ser considerado uma das maiores invenções da humanidade. A sua origem, séculos atrás, se deu naturalmente no seio da sociedade, cumprindo um papel importante no auxílio das transações econômicas voluntárias. Com ele, os indivíduos tornaram-se capazes de poupar seus ganhos excedentes e de adquirirem mais bens e serviços. Até então, cada indivíduo apenas consumia e utilizava aquilo que ele fosse capaz de adquirir por meio da coleta ou da caça. A partir daí, surgiu o escambo, onde as pessoas poderiam trocar os seus bens excedentes diretamente. Porém, com o avanço da produção e manufatura, mais produtos foram surgindo, a prosperidade aumentando e o escambo tornou-se uma prática inviável devido à dificuldade na realização das trocas (SCHIFF, 2012).

Desde então, como meio de padronizar essas unidades de troca em cada localidade ao redor do planeta, o dinheiro assumiu várias formas, desde elementos oriundos da natureza, como as especiarias e os metais, até o atual papel-moeda. Cada um desses bens que servem ou serviram como meio de troca possui as suas vantagens e desvantagens (portabilidade, divisibilidade, peso, facilidade de falsificação, entre outras medidas).

Ao longo dos séculos, cada vez mais os governos foram se apropriando do controle e da emissão de moedas, consequentemente se autofinanciando e aumentando seu poder sobre as sociedades, culminando com a total ruína do sistema monetário antigo e pondo em prática a criação de moeda sob o regime de bancos centrais (ULRICH, 2014). Após as crises econômicas mundiais causadas pela total estatização do dinheiro e da economia, principalmente no século XX e XXI, o ápice da crise mundial de 2008 coincidiu com o surgimento de uma nova tecnologia, a criptomoeda.

A moeda foi reinventada em forma de código computacional. Satoshi Nakamoto, pseudônimo utilizado pelo(s) criador(es), lançou um *white paper*[1] detalhando a criação de um novo tipo de moeda e do sistema de pagamento totalmente descentralizado, chamado de *Bitcoin*, de código aberto, que possuía seu valor determinado pelos indivíduos no mercado e que não seria emitido por nenhuma autoridade central, trazendo de volta às pessoas o total controle do seu dinheiro (ULRICH, 2014). O *Bitcoin* funciona em cima da *Blockchain*, uma tecnologia que é responsável por registrar todos os dados de transações envolvendo a moeda em um livro-razão criptografado e imutável.

Apesar do *Bitcoin* surgir com o conceito *peer-to-peer* (ponto a ponto), eliminando a necessidade de intermediários, após dez anos do seu surgimento, o número de criptomoedas concorrentes aumentou exponencialmente e, assim como ocorreu com outras moedas ao longo do tempo, surgiu a necessidade de realizar trocas entre as mesmas.

As *exchanges* (corretoras) são plataformas *online*, com funcionamento em tempo real que servem como intermediárias transações. Elas organizam as informações de cada negociação em livros abertos.

[1] *https://bitcoin.org/bitcoin.pdf*

Devido à facilidade oferecida pelas corretoras de criptomoedas aliada ao tempo necessário para acompanhar as transações, preços e oportunidades de lucros, foi inevitável o surgimento de ferramentas auxiliares que automatizarem todo o processo de negociação. Tais ferramentas – também denominadas de robôs ou *bots* – facilitam o trabalho dos investidores, tendo em vista que elas podem simular as ações dos mesmos em tempo mais hábil e com capacidade de processamento de informações além do limite humano, além da flexibilidade de trabalhar simultaneamente com um leque de configurações de como agir no mercado.

Todas essas inovações vêm ocorrendo de forma rápida e as *exchanges* ainda estão no processo de adaptação de tais mudanças, desenvolvendo e disponibilizando suas informações para o público, principalmente aqueles que desejam desenvolver aplicações e serviços relacionados.

1.1 PROBLEMA

Essa disponibilização de dados distribuída, em tempo real e com valores precisos é feita por meio de *APIs – Application Programming Interface* ou, em português, Interface de Programação de Aplicativos – um conjunto de padrões e rotinas estabelecidos por um software para serem utilizados por outras aplicações sem que elas envolvam-se com detalhes internos da implementação deste sistema, mas, apenas com os serviços oferecidos.

Por serem serviços *web*, as corretoras de criptoativos seguem um estilo de arquitetura denominado *REST – Representational State Transfer* (Transferência de Estado Representacional) – no desenvolvimento de suas *APIs*. O padrão *REST* é um protocolo difundido mundialmente motivado por seu desempenho, confiabilidade e capacidade de escalabilidade.

A automatização de tarefas relacionadas à operações nas *exchanges* ainda carecem de amadurecimento. A maioria delas ainda estão em processo inicial no desenvolvimento de suas *APIs*, e problemas como: inconsistência de dados, falta de tratamento de erros, falta de flexibilidade na consulta de dados, códigos mal escritos, documentação mal elaborada, atualizações esporádicas, utilização de códigos não oficiais feitos por terceiros, limitações na utilização, entre outros; são comuns no cotidiano dos desenvolvedores, os quais enfretam desafios devido à falta de padronização na disponibilização dos dados e pela alta escassez de materiais de referência, manuais e tutoriais que os auxiliem no consumo desses serviços.

1.2 JUSTIFICATIVA

Neste trabalho, é realizada uma análise comparativa entre a forma de comunicação e os dados disponibilizados por seis exchanges (Binance[2], Bittrex[3], Gate.io[4], Huobi[5], Livecoin[6], e Poloniex[7]) através do estudo de um conjunto de padrões e regras de boas práticas de desenvolvi-

[2] *https://www.binance.com/*
[3] *https://international.bittrex.com*
[4] *https://www.gate.io/*
[5] *https://www.hbg.com/*
[6] *https://www.livecoin.net*
[7] *https://poloniex.com/*

mento de *APIs REST*, aliado à ferramentas que auxiliem na realização desta análise, a fim de criar um documento que sirva como base para os desenvolvedores lidarem com as particularidades e limitações das *exchanges* e que as dificuldades encontradas ao trabalhar com essas *APIs* sejam reduzidas.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Este trabalho tem como objetivo geral efetuar uma análise comparativa entre as principais *APIs* de corretoras de criptomoedas com base em regras, boas práticas e padrões de desenvolvimento de *APIs*, a fim de subsidiar outros desenvolvedores que atuam nesta área.

1.3.2 Objetivos Específicos

Para atender o objetivo geral, foram definidos os seguintes objetivos específicos:

- Definir as regras que serão utilizadas como parâmetro;
- Utilizar ferramentas como o *cURL*, o *Postman* e o navegador para acessar as informações disponibilizadas pelas *APIs*;
- Analisar as informações que foram obtidas com base nos parâmetros especificados;
- Afirmar se as regras foram cumpridas por cada *exchange*;
- Realizar uma comparação entre as análises individuais de cada corretora e documentar o resultado.

2 REFERENCIAL TEÓRICO

2.1 SISTEMAS DISTRIBUÍDOS PONTO A PONTO

As redes *peer-to-peer*(também denominado *p2p* ou ponto a ponto) são compostos de nós - computadores individuais - e, diferente dos sistemas de arquitetura centralizada, têm seus recursos computacionais (armazenamento, dados ou banda) compartilhados diretamente com todos os outros integrantes da rede, sem que haja um membro central que coordena todo o funcionamento. Portanto, permite que todos os nós detenham o mesmo poder, direitos e funções, sendo, simultaneamente, tanto fornecedores quanto consumidores de recursos (DRESCHER, 2018).

Esse tipo de arquitetura é propícia para o desenvolvimento de aplicações de compartilhamento de arquivos, distribuição de conteúdos e proteção de privacidade. A principal característica dos sistemas *p2p* é a descentralização.

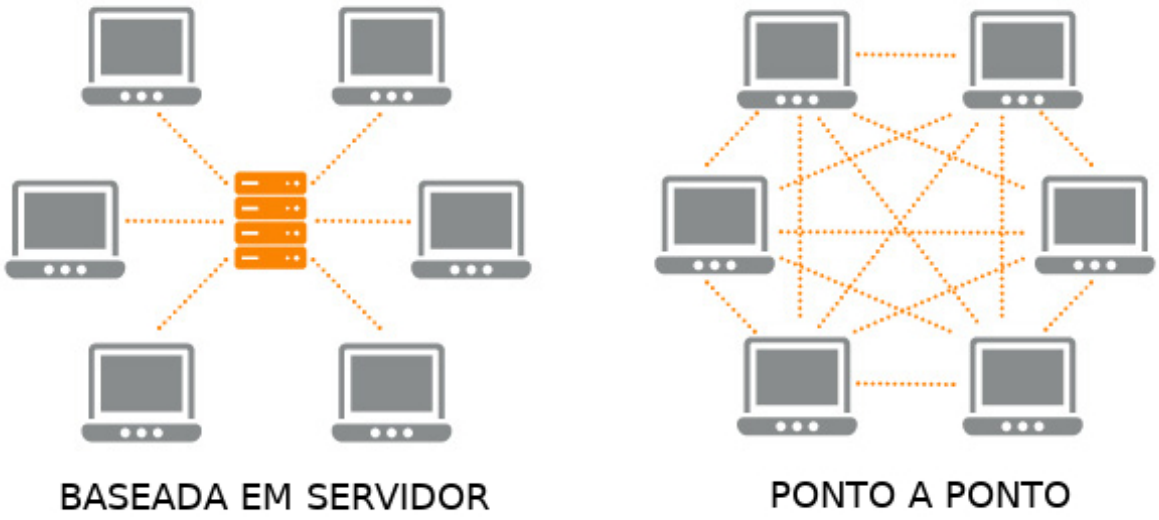

Figura 1 – Rede com a arquitetura centralizada à esquerda e rede ponto a ponto à direita
Fonte: (WOWZA MEDIA SYSTEMS, 2019)

Na arquitetura centralizada, toda a rede fica dependente de um nó central, que fornecerá todos os recursos necessários para o funcionamento da rede. Esse nó detém todo o poder e controle, caso haja algum problema com ele, a rede ficará comprometida por completo. Já nas redes *peer-to-peer*, isso não é um empecilho. Caso algum nó sofra com algum problema ou se desconecte voluntariamente da rede ou perca todas as informações, ninguém será afetado, todos continuarão realizando suas ações normalmente.

Apesar das vantagens, os sistemas ponto a ponto apresentam contrapartidas. Entre elas estão, principalmente, a integridade e confiança. Problemas estes que podem estar relacionados com falhas técnicas, nós mal intenciondos (usuários maliciosos), vulnerabilidade, integridade de informação, entre outros.

2.2 BITCOIN

O *Bitcoin* geralmente é tratado como se fosse apenas uma moeda. Porém, o *Bitcoin* é uma coleção de conceitos e tecnologias responsáveis por formar a base de todo um ecossistema um dinheiro digital. (ANTONOPOULOS, 2014) Dentro dessa variedade de conceitos que podem ser atribuídos a ele, podemos destacar três: (1) Um ativo digital (criptomoeda), (2) uma referência à tecnologia *Blockchain* (livro-razão descentralizado) e (3) o protocolo que é executado sobre a tecnologia *Blockchain* para descrever como os ativos são transferidos (*softwares* que conduzem a transação) (SWAN, 2015).

Todo esse sistema de pagamento de dinheiro digital foi lançado em 2009, sendo desenvolvido com um *design* descentralizado e protegido por uma poderosa criptografia, tornando-o resiliente contra manipulações. (CAETANO, 2015) Os usuários possuem chaves que permitem com que eles provem a sua posse das moedas no decorrer das transações. Tais chaves são sempre armazenadas em uma carteira digital, a qual pode existir no computador do usuário, em sites de transações, em *exchanges*, em *hardwares*, ou em outras plataformas digitais.

2.3 BLOCKCHAIN

Assim, como o *Bitcoin*, o *Blockchain* também abrange um leque de significados, como: (1) uma terminologia para uma estrutura de dados, (2) o nome de um algoritmo, (3) um conjunto de tecnologias e (4) um termo abrangente para sistemas *peer-to-peer* puramente distribuídos com uma área de aplicação comum.

Quando utilizado para nomear uma estrutura de dados, o termo refere-se a vários dados unidos em unidades chamadas de blocos. Tais blocos são conectados uns aos outros de forma encadeada, daí vem o nome *Blockchain* (cadeia de blocos, em tradução livre). Quando atribuído a um algoritmo, o significado refere-se a um conjunto de instruções que lidam com o conteúdo de muitas estrutura de dados *Blockchain* em um sistema *p2p*. Ao se referir a um conjunto de tecnologias, o termo inclui a estrutura de dados, o algoritmo, tecnologias de criptografia e segurança os quais podem ser utilizados para prover integridade em sistemas puramente distribuídos ponto a ponto. E, diferente do significado anterior, a quarta atribuição ao termo é referente à um sistema distribuído como um todo, não unicamente a uma unidade de *software* que faz parte desse tipo de sistema. (DRESCHER, 2018)

O desenvolvimento desta tecnologia foi um avanço fundamental na ciência da computação, juntando cerca de quarenta anos de pesquisas em criptografia com vinte anos de pesquisas em moedas criptográficas. (SWAN, 2015) O Blockchain resolve um problema de longa data chamado “gasto duplo”. Este problema ocorre quando duas transações são aceitas com um montante que excede o valor antes disponível para gasto, ou seja, aquela quantia foi usada mais de uma vez.

Até o desenvolvimento do *Blockchain*, o dinheiro digital não era escasso (assim como todos os outros recursos digitais), podendo ser copiados e replicados *ad infinitum*, e não havia

uma maneira de confirmar se aquele recurso já havia sido gasto sem que houvesse uma terceira parte envolvida para intermediar e realizar a sincronização entre todas as transações (SWAN, 2015).

Para resolver o gasto duplo, o *Blockchain* provê um mecanismo de confirmação e um livro-razão universal distribuído para que todos os nós (pontos) estejam informados e atualizados ao longo de cada transação. Cada informação nova adicionada na cadeia de blocos é armazenada em ordem cronológica, assim, fazendo com que o rastreamento seja feito de maneira simples. A cada 10 minutos um novo grupo de transações – ou seja, um bloco – é adicionado ao livro e todos os nós possuirão uma cópia do mesmo. Caso alguém tente usar o mesmo recurso mais de uma vez, será impossível, pois uma vez que a primeira operação dela foi iniciada, a mesma vai para um *pool* de transações não confirmadas. Apenas a primeira das duas transações será confirmada e verificada pelos mineradores, enquanto a segunda será classificada como inválida e não terá confirmações suficientes para ser validada; e mesmo que as duas transações sejam feitas ao mesmo tempo, a que tiver mais confirmações dos mineradores (no mínimo seis, para que seis outros blocos sejam adicionados no topo do bloco que está sendo verificado) será a aceita.

A imagem abaixo mostra os passos para a realização de uma transação com *Blockchain*. Primeiro, o usuário requisita uma transação. Em seguida, essa mesma requisição é transmitida para a rede, que será responsável por validar ou rejeitar o pedido de transação. Após isso, caso seja validada, ela é adicionada ao atual bloco de transações, o qual será encadeado com os outros blocos mais antigos, assim, confirmando a transação.

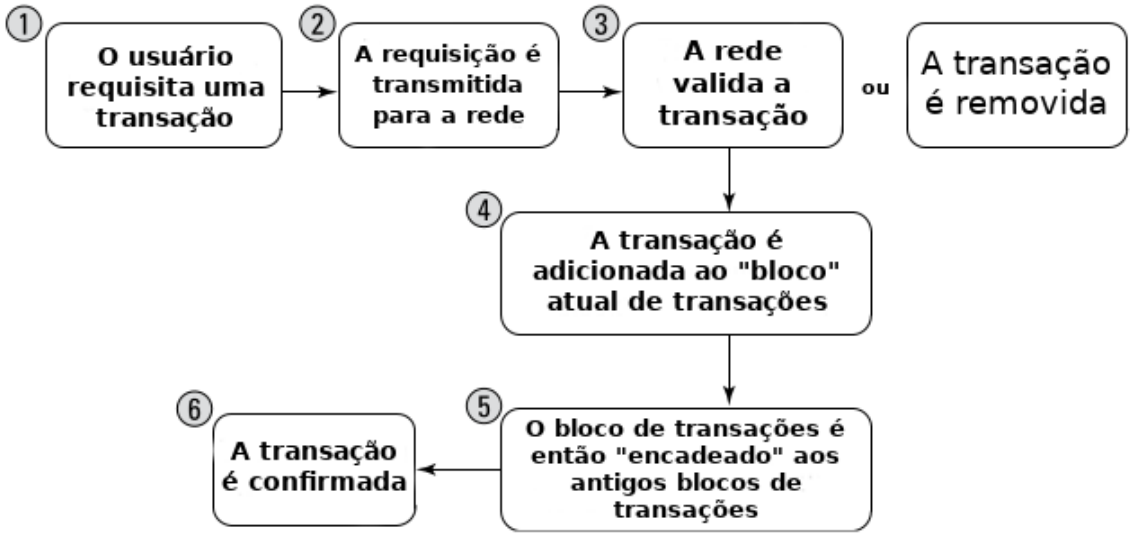

Figura 2 – Funcionamento do Blockchain
Fonte: (LAURENCE, 2017)

A relação entre os sistemas *p2p* e o *Blockchain* é que o último serve para prover manter a integridade do primeiro sem que haja intermediação (DRESCHER, 2018).

2.4 EXCHANGES

As exchanges de criptoativos são plataformas digitais, com funcionamento em tempo real que servem como intermediárias na compra, venda e troca dos mesmos, a fim de facilitar o processo de aquisição. Elas costumam cobrar taxas nessas transações e organizar as informações de cada negociação em livros abertos. As exchanges são intermediários optativos, já que, as criptomoedas podem funcionar de forma independente sem a necessidade de um intermediário envolvido no processo.

As seis *exchanges* abordadas neste trabalho foram escolhidas com base no seu volume de transação e no nível de maturidade de sua *API*. O volume é definido pela soma de todos os pares de mercado (*market pairs*[1]) reportado pela corretora nas últimas vinte e quatro horas.

As informações a respeito do volume das corretoras são encontradas no *Coin Market Cap*[2], um *site* que disponibiliza dados e gráficos em tempo real tanto das *exchanges* quanto das criptomoedas existentes no mercado. No momento da escolha, as corretoras aqui analisadas estavam presentes entre as vinte primeiras posições no *Coin Market Cap*.

2.5 HTTP

Projetado no início da década de 1990, o *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) é um protocolo de comunicação utilizado na transferência de documentos na *Internet*. Ele garante a integridade dos dados transmitidos durante a comunicação. É por meio dele que as informações oriundas de servidores *web* chegam de forma rápida, conveniente e confiável aos internautas, nos seus respectivos navegadores. O *HTTP* segue o modelo de comunicação cliente-servidor, o qual o cliente inicia uma conexão com o servidor, realiza requisições e aguarda até receber alguma resposta.

Os servidores *web* – também conhecidos como servidores *HTTP* – são responsáveis por armazenar e prover todos os tipos de recursos *web*: arquivos de texto, arquivos *HTML*, arquivos de multimídia, etc.. O *client* requisita ao *server* o conteúdo desejado por meio de *HTTP requests* (requisições), e os servidores retornam os dados por meio de *HTTP responses* (respostas) (GOURLEY D. E TOTTY, 2002), como mostrado na figura 4.

Além das características citadas, anteriormente, este protocolo é *stateless*, ou seja, cada requisição é independente, mesmo que venham a ocorrer de forma simultânea e na mesma conexão cliente-servidor. Para o *server*, cada *request* é diferente, ele não guarda o estado das requisições anteriores, toda a informação é perdida.

---

[1] *Um market pair é a cotação de duas moedas diferentes, com o valor de uma sendo cotado contra o valor de outra. O par é composto por uma moeda base e outra moeda de cotação.*

[2] *https://coinmarketcap.com/rankings/exchanges/*

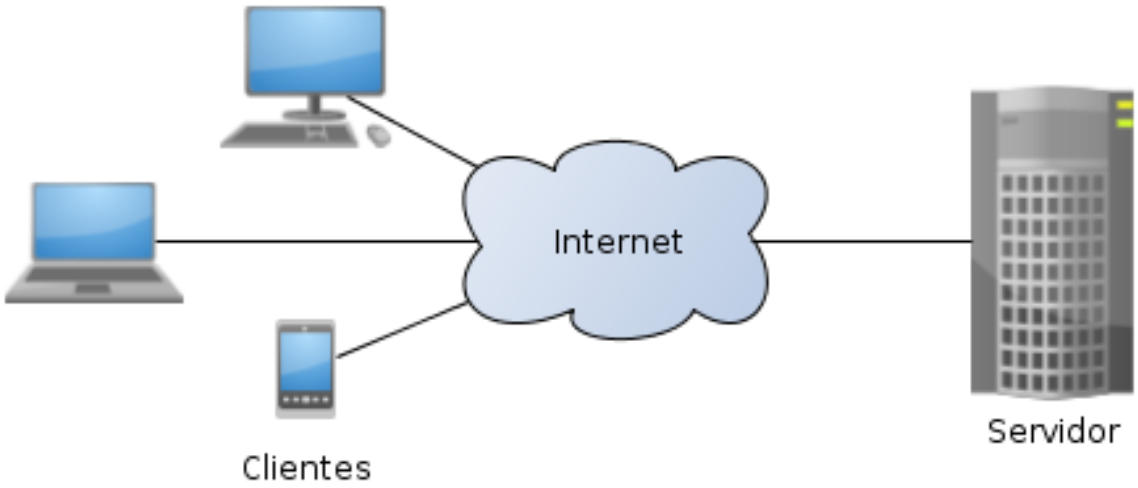

Figura 3 – Modelo de arquitetura cliente-servidor
(WIKIPEDIA, A ENCICLOPEDIA LIVRE, 2013)

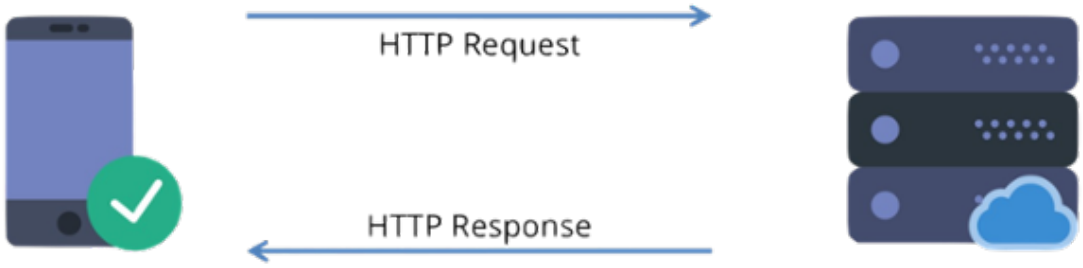

Figura 4 – Funcionamento de uma requisição HTTP
Fonte: (HERNANDEZ, 2017)

2.5.1 REST

REST é um acrônimo para *Representational State Transfer* (Transferência de Estado Transacional), um estilo arquitetural para sistemas *web* distribuídos a fim de facilitar a comunicação entre eles. Foi apresentado em 2000 por Roy Fielding, um dos autores da especificação do *HTTP*, em sua tese de doutorado[3]. Nela, Roy sugeriu um conjunto de princípios, regras e *constraints* que permitem o desenvolvimento de aplicações com interface bem definidas.

As aplicações desenvolvidas que seguem essa arquitetura são denominadas *RESTful*, e seis princípios devem ser respeitados na implementação das mesmas:

- **Cliente-servidor**: a portabilidade da interface do usuário e a escalabilidade do sistema são melhoradas quando há uma separação do cliente, responsável pela interface, do servidor, que armazena as informações;

- **Stateless**: informações de sessão do usuário fica a cargo do cliente. Cada requisição para

[3] *https://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm*

o servidor deve conter todas as informações necessárias para que ele entenda o *request*;

- **Armazenamento em cache**: requer que os dados em uma resposta sejam rotulados implícita ou explicitamente se podem ou não serem armazenados em *cache*. Caso positivo, o cache do cliente poderá utilizar essas informações retornadas do servidor em outras requisições futuras;

- **Interface uniforme**: a fim de simplificar a arquitetura de todo o sistema, o *REST* busca obter uma interface uniforme definindo quatro *constraints*: identificação dos recursos, manipulação dos recursos através de representações, mensagens auto descritivas e hipermídia como motor do estado da aplicação;

- **Sistemas em camadas**: este modelo, também conhecido como *layered systems*, permite que uma arquitetura seja composta de camadas hierárquicas, restringe o comportamento dos componentes de forma que cada um deles não possa acessar outras camadas além da qual eles estão interagindo no momento;

- **Código sob demanda**: este último princípio é optativo. Geralmente o usuário enviará as representações estáticas de recursos (*JSON* ou *XML*), porém, caso seja necessário, é possível retornar o código fonte executável para suportar parte da aplicação que está sendo desenvolvida.

2.6 API

Application Programming Interface – Interface de Programação de Aplicações, em tradução livre – é um conjunto de definições, protocolos e ferramentas para o desenvolvimento e integração de *softwares* de aplicações. Essa interface permite que o sistema possa se comunicar com produtos e serviços oferecidos por outras aplicações, dessa forma, simplificando e tornando mais flexível o processo de construção do sistema, consequentemente, gerando economia de tempo e de recursos.

Com as *APIs*, a aplicação deixa disponível recursos, serviços e informações sem que haja uma perda na segurança e no controle, já que o acesso e as permissões serão especificadas pela equipe de desenvolvimento. Nos últimos anos, o *REST* vem sendo o principal protocolo de padronização de *APIs*.

2.6.1 *REST API: Design Rulebook*

É o principal material utilizado neste trabalho como base para realizar as análises das *APIs*. Segundo (MASSÉ, 2016), as regras servem para ajudar os desenvolvedores a construir *RESTful APIs* consistentes que podem ser desfrutadas pelos usuários. Algumas delas tornaram-se padrões, de fato, outras era implícitas no protocolo *HTTP*. Tais regras podem ser utilizadas

por completo ou de forma separada, dependendo do contexto do serviço, e elas servem para responder algumas questões como:

- Quando os caminhos da *URI* devem ser nomeados com substantivos no plural?
- Quais métodos de requisição devem ser utilizados para atualizar o estado dos recursos?
- Como implementar operações diferentes do *CRUD* nas *URIs*?
- Qual é o código de *status HTTP* apropriado para um determinado cenário?
- Como gerenciar diferentes versões de uma representação do estado de um recurso?
- Como estruturar um *hyperlink* em *JSON*?

2.7 TRABALHOS RELACIONADOS

2.7.1 *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*

Este livro é um guia de como desenvolver, gerenciar e manter *APIs* em sistemas corporativos. O autor aborda questões de segurança, ciclo de vida, documentação, padrões de projeto, controle de versão, monetização, *analytics* e estratégias para testes de performance e testes de carga.

Na seção de seção de *design* da interface da *API*, princípios básicos do *REST*, os verbos, os *status codes*, a estrutura, convenções e boas práticas dos identificadores de recursos *URIs*, versionamento, e outros detalhes importantes e necessários no projeto de construção de uma *API*.

Segundo (DE, 2017), as *APIs* devem ser desenvolvidas para que tenham um longo ciclo de vida. Qualquer mudança acarreta em riscos de funcionamento para o cliente que a consome. Frequentes mudanças geralmente deixam usuários comuns e desenvolvedores frustrados. Utilizar padrões robustos na construção de *APIs* ajuda a comunidade de desenvolvedores e pode salvar a companhia de ter despesas financeiras.

2.7.2 *REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development*

Assim como o trabalho anterior, neste livro o (DOGLIO, 2018) mostra todos os passos necessários para se desenvolver uma *API* e lançá-la em um ambiente de produção. Nele, é feita uma introdução ao *REST* e como desenvolver uma *RESTful API* na linguagem *Javascript* utilizando o *framework Node.js*. A diferença encontra-se na abordagem das boas práticas de desenvolvimento de *APIs*. O livro traz o conteúdo de forma mais superficial, não se aprofundando ou expondo os padrões minuciosamente. Por outro lado, o autor abordar o conteúdo didaticamente.

2.7.3 *API Design for C++*

Este livro trata de forma mais profunda a construção de *APIs* elegantes e robustas. Criar interfaces de alta qualidade é, portanto, uma habilidade essencial de engenharia e o foco central deste trabalho. (REDDY, 2011)

Para auxiliar na performance, o autor utiliza a linguagem de programação *C++*, que permite que a aplicação seja processada até mesmo em unidades de processamento gráfico (GPU). Alguns componentes para aumentar o desempenho das *APIs* são considerados, entre eles: velocidade do tempo de compilação, velocidade do tempo de execução, sobrecarga da memória em tempo de execução, tamanho da aplicação e o tempo de inicialização.

Além do desempenho da aplicação, o livro aborda os diferentes tipos de *patterns* de implementação e possui uma seção que trata da qualidade de uma *API*, descrevendo as características que uma boa aplicação deve ter, entre elas: modelagem do domínio do problema, esconder os detalhes de implementação, ser fácil de utilização, ser testada, documentada e estável.

3 ANÁLISE DAS *APIS* DAS *EXCHANGES*

Neste capítulo são realizadas todas as análises das *APIs* das corretoras de criptomoedas, baseando-se em trinta e três regras retiradas do livro *REST API: Design Rulebooks* e nas limitações de uso definidas pelas próprias *exchanges* na documentação de suas respectivas *API*.

3.1 FERRAMENTAS UTILIZADAS

Nesta pesquisa foram usadas algumas ferramentas para auxiliar no processo de consulta e análise das requisições e respostas das *APIs*.

3.1.1 Postman

O *Postman* é uma aplicação desenvolvida com a finalidade de realizar diversos tipo de testes em *APIs RESTful*. Através da sua interface, é possível enviar requisições *HTTP*, analisar os resultados, consultar informações mais detalhadas e ter uma visão mais aprofundada do funcionamento dos serviços que estão sob consulta. Esta plataforma também possibilita que os desenvolvedores documentem as suas *APIs* e possam planejar e escrever casos de testes.

Na imagem abaixo temos um exemplo do uso da ferramenta neste trabalho. Nele, inserimos a *URI* e selecionamos o verbo *HTTP* e realizamos a requisição. O *software* exibirá na sua interface a resposta da requisição formatada, o cabeçalho do *response*, informações do tempo de duração e do tamanho da resposta, *status ode*, entre outros.

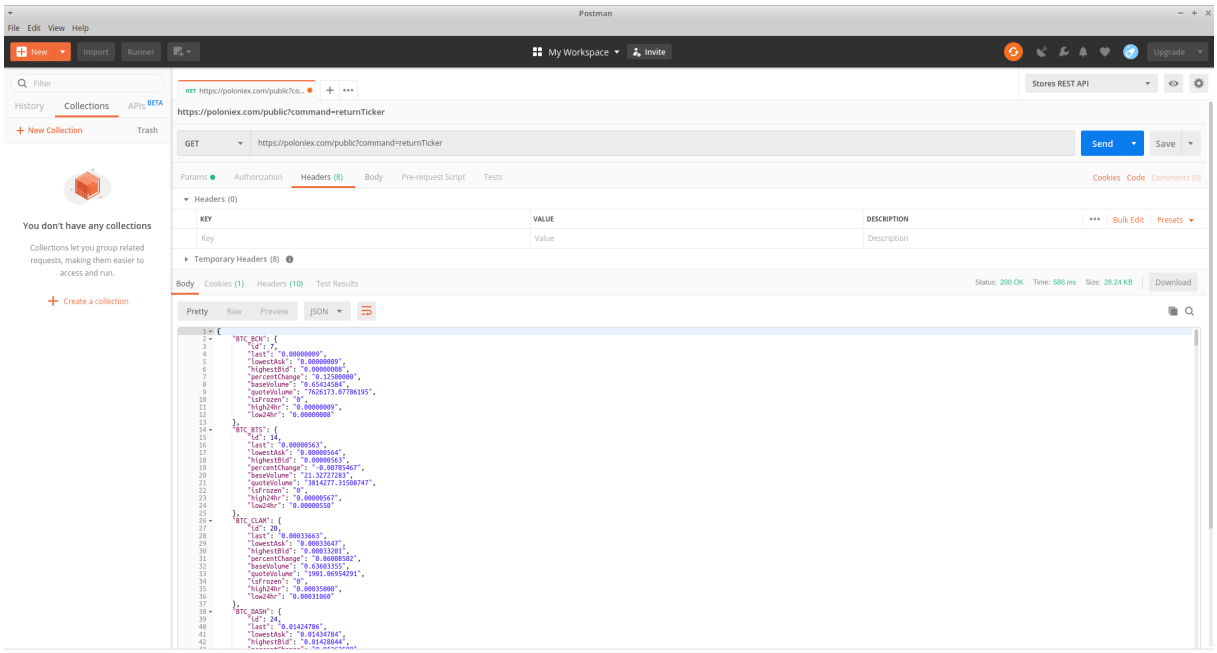

Figura 5 – Interface do Postman

3.1.2 Google Chrome

O navegador Google Chrome disponibiliza em seu próprio *software* as "Ferramentas de desenvolvedor", as quais permitem ao usuário analisarem uma gama de recursos e conteúdos de uma página na *Internet*. Com elas, é possível ver e editar o código-fonte da página, acessar os arquivos e recursos da aplicação, visualizar as informações e dados enviadas na requisição e recebidos na resposta além de mensagens de erros, monitorar o tráfego *HTTP*, depurar códigos desenvolvidos na linguagem *JavaScript*, acessar informações do uso de memória e fontes de latência, entre outros.

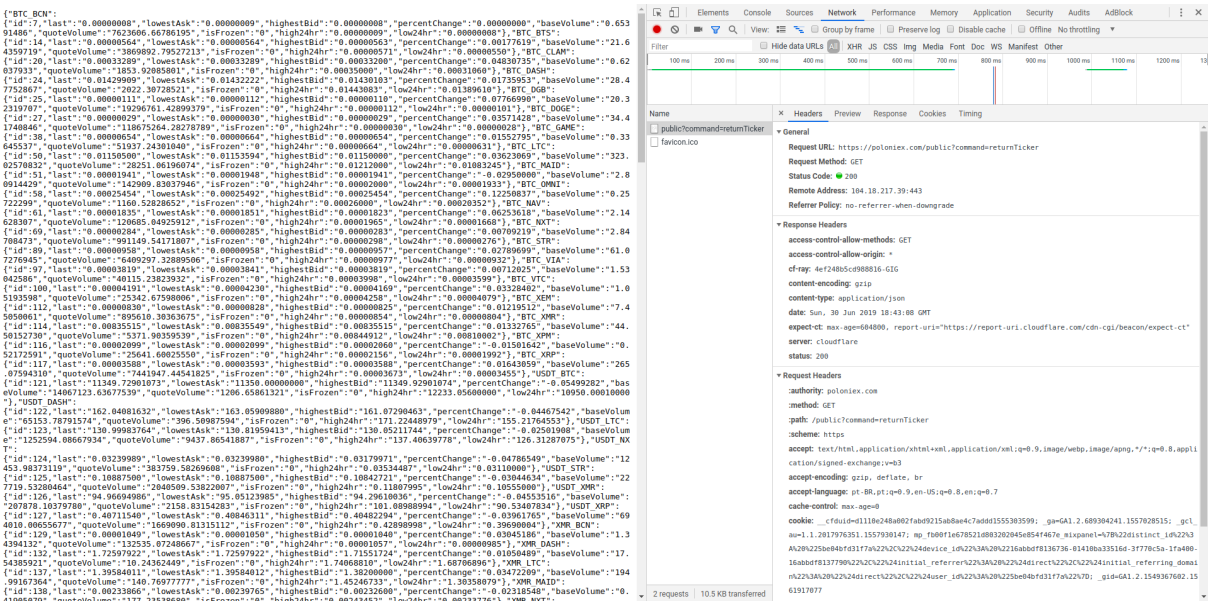

Figura 6 – Interface das Ferramentas de desenvolvedor

No exemplo acima é mostrado como uma requisição é feita pelo navegador e como as suas informações são dispostas na interface. No lado esquerdo a resposta é exibida ao usuário, já o lado oposto é composto pela ferramenta de desenvolvedor, com detalhes da requisição e da reposta, informações gerais - *URL*, método *HTTP*, *status code*, etc. -, cabeçalhos do *request* e do *response*, além da possibilidade de ver possíveis mensagens de erro na aba *Console*, informações a respeito da rede na aba *Network* e dados de memória e performance.

3.1.3 CURL

O *Client URL* é uma ferramenta de linha de comando, podendo também ser utilizada em *scripts*, presente nos Sistemas Operacionais baseados em *Unix*, e são destinados a verificar a conectividade da *URL*, podendo ser utilizado também para realizar transferência de dados. Ele suporta vários protocolos utilizados na *Internet*, dentre eles o *HTTP*.

Em determinadas *exchanges*, para acessar as rotas privadas é necessário que haja um código criptografado que é gerado através da chave privada da *API*, do comando desejado e da data e hora atual em milissegundos (*nonce*), utilizando o algoritmo de criptografia *hash SHA-512*.

```
echo -n "command=returnCompleteBalances&nonce=154264078495300" | \
openssl sha512 -hmac $API_SECRET

curl -X POST \
     -d "command=returnCompleteBalances&nonce=154264078495300" \
     -H "Key: 7BCLAZQZ-HKLK9K6U-3MP1RNV9-2LS1L33J" \
     -H "Sign: 2a7849ecf...ae71161c8e9a364e21d9de9" \
     https://poloniex.com/tradingApi
```

Figura 7 – Uso do cURL

3.2 REGRAS SELECIONADAS

3.2.1 Modelagem de identificadores com *URIs*

O Identificador de Recursos Universal, ou *URI* - Unified Resource Language - é um elemento presente nas *APIs REST*. O mesmo é utilizado para endereçar recursos. Cada recurso deve possuir um nome significativo para que o mesmo seja identificado. Modelá-los é uma das mais importantes tarefas para o sucesso de uma *API*, sendo responsável por deixá-la mais intuitiva e de fácil manuseio para os seus usuários (DE, 2017).

3.2.1.1 Formato da *URI*

O formato de uma *URI* deve seguir a seguinte estrutura abaixo:

esquema "://"**autoridade** "/"**caminho** ["?"**consulta**] ["#"**fragmento**]

- **Esquema:** É o espaço utilizado para identificar o protocolo que está sendo utilizado. Por exemplo: *HTTP, HTTPS, FTP*, etc;
- **Autoridade:** Representa a resolução do *Domain Name System* do servidor em que a aplicação está sendo executada. Pode ser composta pelo *hostname* ou pelo endereço de *IP*, opcionalmente com o número da porta ou com credenciais de acesso;
- **Caminho:** Determina uma sequência de segmentos dipostos de forma hierárquica e separados por uma barra (/);
- **Consulta:** São informações adicionais, não hierárquicas, de identificação. Sempre são precedidas de uma interrogação;
- **Fragmento:** São separados por cerquilhas (#) e direcionam para recursos secundários dentro dos recurso primário, o qual é identificado pela autoridade.

Regra 1 - O separador (/) deve ser usada para indicar relacionamento hierárquico.

- **Binance:** apesar de seguir a regra, algumas rotas poderiam ter uma hierarquia organizada de forma mais eficiente. Por exemplo: as rotas */api/v3/openOrders* e */api/v3/allOrders* deveriam ser atreladas a um *endpoint /orders/*, que retornaria todas as ordens, e com a possibilidade eu filtrar a ordem pelo *status* (aberta, finalizada, cancelada, etc.) em uma *query*.
- **Bittrex:** utiliza a regra de forma coerente.
- **Poloniex:** não trabalha com hierarquia, todos os comandos são executados por meio de uma *query* ou por meio do corpo da requisição utilizando *JSON*.
- **Livecoin:** a hierarquia deveria ser melhor estruturada. Por exemplo: para as rotas */exchange/buylimit* e */exchange/selllimit*, o correto seria ter um caminho */exchange/limit* e eu escolher o tipo (*buy* ou *sell*) por consulta, por meio do corpo da mensagem via *JSON* ou, até mesmo, criando mais duas rotas */exchange/limit/buy* e */exchange/limit/sell*.
- **Gate.io:** assim como na Binance e na Livecoin, o problema com a organização da hierarquia se repete. Rotas como */private/cancelOrders* e */private/cancellAllOrders* poderiam ser provenientes da hierarquia */private/orders*.
- **Huobi:** utiliza a regra de forma coerente.

Regra 2 - O separador (/) não deve ser incluída no final da *URI*: o acréscimo deste elemento não adiciona nenhum conteúdo semântico e pode causar confusão.

Todas as corretoras cumprem esta regra.

Regra 3 - Hífens (-) devem ser utilizados para melhorar a legibilidade da *URI*.

- **Binance:** utiliza o padrão *camelCase*, onde a primeira palavra inicia com letra minúscula e as demais com letra maiúscula.
- **Bittrex:** não separa as palavras.
- **Poloniex:** utiliza o padrão *camelCase*.
- **Livecoin:** não possui um padrão, em determinados *endpoints* faz a separação com sublinha (_), enquanto outros são separados com o padrão *camelCase*.
- **Gate.io:** não tem um padrão definido, alterna entre sublinhas e *camelCase*.
- **Huobi:** padronizada com o *camelCase*.

Regra 4 - Sublinhas (_) não devem ser usadas nas *URIs*: geralmente, aplicações de visualização de texto utilizam a sublinha para indicar que aquele elemento é clicável. Além

disso, dependendo da fonte utilizada na aplicação, a sublinha pode ficar escondida ou de difícil visualização.

Como mostrado na regra de número 3, as corretoras Livecoin e Gate.io fazem uso das sublinhas, dessa forma ambas não cumprem a regra.

Regra 5 - Letras minúsculas devem ser preferidas em caminhos da *URI*: A *RFC* 3986[1] define que as *URIs* são *case-sensitive*, ou seja, diferenciam letras minúsculas e maiúsculas nos componentes do Identificador de Recurso Universal (com exceção do esquema e da autoridade).

Apenas a Bittrex utiliza letras minúsculas em todas as suas rotas.

Regra 6 - Extensões de arquivos não devem ser inclusas nas *URIs*. Todas as *exchanges* cumprem esta regra nas suas *APIs*.

3.2.1.2 Modelagem de autoridade da *URI*

Regra 7 - Nomes de subdomínios consistentes devem ser utilizados: os nomes do domínio de alto nível e do primeiro subdomínio (ex.: ifrn.edu.br) de uma *API* deve identificar o proprietário do serviço ofertado. O nome completo deve incluir um subdomínio chamado **api**, como na estrutura a seguir.

https://api.ifrn.edu.br

- **Binance:** cumpre a regra.
- **Bittrex:** cumpre a regra.
- **Poloniex:** não segue o padrão *api.subdominio*, além de possuir duas ramificações na *URI*, uma para a api pública (*poloniex.com/public*) e outra para a privada (*poloniex.com/tradingApi*).
- **Livecoin:** cumpre a regra.
- **Gate.io:** tem quatro subdomínios, dados para os dados públicos (*data.gateio.co/api2/1* e *data.gateio.io/api2/1*) e mais dois para informações privadas (*api.gateio.co/api2/1/private* e *api.gateio.io/api2/1/private*).
- **Huobi:** cumpre a regra.

3.2.1.3 Resumo

A Huobi foi a *exchange que mais seguiu os padrões de modelagem de URI*, cumprindo cinco regras e descumprindo apenas uma. Em seguida temos a Bittrex e a Binance cumprindo cinco e quatro regras, respectivamente. As demais corretoras não tiveram um bom desempenho nessa análise, cumprindo menos de quatro regras.

[1] *https://www.ietf.org/rfc/rfc3986.txt*

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 1	Parcialmente	Sim	Não	Parcialmente	Parcialmente	Sim
Regra 2	Sim	Sim	Sim	Sim	Sim	Sim
Regra 3	Não	Não	Não	Não	Não	Não
Regra 4	Sim	Sim	Sim	Não	Não	Sim
Regra 5	Parcialmente	Não	Parcialmente	Parcialmente	Parcialmente	Parcialmente
Regra 6	Sim	Sim	Sim	Sim	Sim	Sim
Regra 7	Sim	Sim	Não	Sim	Não	Sim

Tabela 1 – Cumprimento das regras de modelagem de identificadores com *URI* por *exchange*

É notável que nenhuma das *exchanges* seguiu a terceira regra. Todas optaram por utilizar o padrão *camelCase* ao invés de utilizar a separação por hífens. Apesar disso, essa violação não causará impacto na utilização das *APIs*.

3.2.2 Modelagem de recursos

A modelagem de recursos é um processo semelhante à modelagem de dados em bancos relacional. Ela é ajuda a fixar alguns conceitos-chave no desenvolvimento da *API*. É de demasiada importância escolher os recursos certos e modelá-los minuciosamente, para que os consumidores da aplicação possam receber as funcionalidades, comportamentos e manutenções desejadas (SUBRAMANIAM, 2014).

De forma similar aos padrões de projeto (*design patterns*), o projeto de recursos de uma *API* é composto por quatro **arquétipos de recursos** (documento, coleção, armazenamento e controlador), os quais ajudam a comunicação se tornar mais consistente entre as estruturas e comportamentos comuns.

- **Documento (*document*):** representa um único recurso dentro de uma coleção. É um conceito singular semelhante a uma instância de um objeto ou registro em um banco de dados;

- **Coleção (*collection*):** equivale a um diretório de recursos gerenciado pelo servidor. Recursos adicionais podem ser inseridos pelos usuários caso seja permitido pela coleção;

- **Reserva (*store*):** diferente das coleções, uma reservas é um repositório de recursos gerenciado pelo cliente;

- **Controlador (*controller*):** controladores são equivalentes à funções executáveis, possuindo parâmetros, valores de retorno, dados de entrada e de saída.

3.2.2.1 Modelagem do caminho da *URI*

Regra 8 - Substantivo no plural devem ser utilizados para nomes de coleções.

Todas as corretoras seguem esta regra, porém, a Poloniex utilizam um padrão de nomeação das rotas que retornam coleções semelhante a uma função na programação, por exemplo: *returnBalances, returnOpenOrders*.

Regra 9 - Um verbo ou uma frase verbal deve ser utilizada para nome de controladores.

A Binance é a única que, em algumas rotas, como a */api/v3/order*, não há um verbo ou uma frase verbal para executar controladores; no exemplo citado, para criar ou remover uma ordem.

Regra 10 - Nomes de funções *CRUD* não devem ser utilizados na *URI*: identificadores devem ser utilizados para indicar apenas recursos, nunca para indicar que funções de criar, ler, atualizar e deletar conteúdos (*CRUD - create, read, update, delete*) estão sendo utilizadas.

Todas as corretoras seguiram esta regra no desenvolvimento de suas *APIs*.

3.2.2.2 Resumo

Todas as corretoras cumpriram as regras de modelagem de recursos, com exceção da Binance que cumpre parcialmente a nona regra. Tal cumprimento ocorre devido ao fato das *APIs* das *exchanges* compartilharem funções, recursos e informações semelhantes.

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 8	Sim	Sim	Sim	Sim	Sim	Sim
Regra 9	Parcialmente	Sim	Sim	Sim	Sim	Sim
Regra 10	Sim	Sim	Sim	Sim	Sim	Sim

Tabela 2 – Cumprimento das regras de modelagem de recursos por *exchange*

3.2.3 Modelagem de consulta da *URI*

O componente de consulta (*query*) é opcional em uma *URI*. O mesmo contém um conjunto de parâmetros para serem interpretados como uma variação ou derivação do recurso determinado no caminho hierárquico do identificador. Ele fornece uma interação adicional entre o usuário e a *API*, como a procura por termos específicos e filtros.

Regra 11 - O componente de consulta de uma *URI* pode ser utilizado para filtrar coleções ou reservas: no exemplo abaixo temos uma forma de utilizar as *queries* para filtrar a busca por uma coleção através de um valor específico.

https://api.ifrn.edu.br/alunos?curso=informatica

A regra foi cumprida por todas as *exchanges*.

Regra 12 - O componente de consulta de uma *URI* dever ser usado para paginar coleções ou *stores*: o cliente deve ser capaz de utilizar as *queries* para retornar coleções paginadas e resultados de reservas com os parâmetros *pageSize* e *pageStartIndex*, indicando a quantidade máxima de conteúdo desejado por consulta e o índice do primeiro elemento.

https://api.ifrn.edu.br/alunos?pageSize=10&pageStartIndex=20

No exemplo mostrado acima, a *API* retornará dez elementos, com o primeiro sendo equivalente ao vigésimo em uma consulta sem filtros de paginação, ou seja, o índice inicial será vinte.

Das seis corretoras, apenas a Binance faz uso do *limit* para delimitar a quantidade de elementos retornados, mas não há *queries* para paginar os resultados.

3.2.3.1 Resumo

A décima primeira regra foi cumprida por todas as *exchanges*, porém, a última dessa seção, não o foi.

Há alguns motivos para os desenvolvedores dessas *APIs* não se preocuparem com paginações, entre eles: a limitação geralmente ocorre utilizando-se outras informações - nome da moeda, a data e hora, etc. -, e, também, porque em algumas rotas os dados retornados em tempo real podem variar a cada milésimo de segundo e, dependendo da finalidade de quem consome a *API*, o retorno de apenas uma amostra de todo o conjunto de dados pode interferir no resultado desejado, dessa forma, é melhor e mais fácil disponibilizar toda a massa de dados.

Regras	**Binance**	**Bittrex**	**Poloniex**	**Livecoin**	**Gate.io**	**Huobi**
Regra 11	Sim	Sim	Sim	Sim	Sim	Sim
Regra 12	Não	Não	Não	Não	Não	Não

Tabela 3 – Cumprimento das regras de modelagem de consulta de *URI* por *exchange*

3.2.4 Modelagem de interação com *HTTP*

RESTful APIs suportam todos os elementos e aspectos do protocolo *HTTP* na sua versão 1.1, como os métodos de requisição - popularmente conhecidos como verbos *HTTP* -, códigos de resposta e cabeçalhos.

3.2.4.1 Métodos de requisição

O *HTTP* é composto por oito verbos:

- ***GET:*** utilizado para recuperar dados e informações;
- ***POST:*** tem como função enviar dados para o servidor;
- ***PUT:*** substitui todas as informações do recurso destinado com o conteúdo submetido;
- ***PATCH:*** similar ao *PUT*, a diferença é que o *PATCH* faz uma atualização parcial de um recurso, ou seja, apenas alguns atributos do recurso em questão podem ser especificados para atualização;
- ***HEAD:*** possui a mesma funcionalidade do método *GET*, porém, o servidor deve retornar na resposta apenas o cabeçalho e o *status* e não o corpo da mensagem;

- ***DELETE:*** remove a representação do recurso alvo;
- ***OPTIONS:*** descreve as opções de comunicação com o recurso alvo;
- ***CONNECT:*** inicia a comunicação bidirecional com o recurso alvo, estabelecendo um túnel com o servidor;
- ***TRACE:*** realizar um teste, enviando uma mensagem pelo caminho até o recurso alvo e o servidor respondendo a mensagem recebida.

Regra 13 - *GET* deve ser utilizada para recuperar a representação de um recurso.

Todas as *exchanges* seguem a regra, a única observação é a respeito da Bittrex, que utiliza apenas o *GET* para todas as rotas.

Regra 14 - *GET* e *POST* não devem ser usados no lugar de outros métodos de requisição.

- **Binance:** utiliza corretamente ambos os verbos.
- **Bittrex:** utiliza o *GET* para todas as requisições.
- **Poloniex:** usa o *GET* corretamente, mas o *POST* é utilizado no lugar do *DELETE*.
- **Livecoin:** usa o *GET* corretamente, porém, o *DELETE* é substituído pelo *POST*.
- **Gate.io:** usa o *GET* corretamente, mas o *POST* é utilizado no lugar do *DELETE*.
- **Huobi:** usa o *GET* corretamente, porém, o *DELETE* é substituído pelo *POST*.

Regra 15 - *POST* deve ser usado para criar novos recursos em uma *collection*.

A única exceção à esta regra é a Bittrex, pelo motivo citado nas duas últimas regras.

Regra 16 - *POST* deve ser utilizado para executar controladores.

A Bittrex é a única que não cumpriu.

Regra 17 - *DELETE* deve ser utilizado para remover um recurso.

Apenas a Binance cumpriu a regra. Todas as outras atribuíram esta funcionalidade ao *GET* ou ao *POST*.

3.2.4.2 *Status codes* de resposta

O protocolo *HTTP* define uma gama de códigos de estado para que sejam usados a fim de transmitir para o cliente os resultados de uma requisição. Os *status codes* são subdivididos em cinco categorias:

- **Informacionais:** retorna informações de transferência a nível de protocolo. Sempre iniciam com o número 1;
- **Sucesso:** indica que a requisição foi aceita com sucesso. Iniciam com o número 2;

- **Redirecionamento:** informa ao cliente que ele deve executar mais ações para que sua requisição seja completada. Os códigos desta categoria começam com o número 3;
- **Erro no cliente:** indica que o erro na resquisição aconteceu do lado do cliente. Sempre iniciam com o número 4;
- **Erro no servidor:** indica que o erro na resquisição aconteceu do lado do servidor. Sempre iniciam com o número 5;

Regra 18 - 200 (*"Ok"*) deve ser usado para indicar sucesso não específico.

Regra cumprida pelas *APIs* de todas as *exchanges*.

Regra 19 - 200 (*"OK*) não deve ser utilizado para comunicar erros no corpo da resposta.

- **Binance:** respeita a regra, retorando o código de erro equivalente ao problema e uma mensagem com um código de erro próprio e uma descrição.
- **Bittrex:** em alguns casos retorna o código 200.
- **Poloniex:** retorna 200, violando a regra.
- **Livecoin:** retorna 200, apesar de informar no corpo da *response* um código de erro próprio.
- **Gate.io:** retorna 200, apesar de informar no corpo da *response* um código de erro próprio.
- **Huobi:** retorna 200, apesar de informar no corpo da *response* um código de erro próprio no formato texto.

Regra 20 - 204 (*"No Content"*) deve ser utilizado quando o corpo da resposta está intencionalmente vazio.

Todas retornam o *status code* 200. Apesar da Bittrex ser a única que não retorna uma mensagem vazia, ao contrário das outras, apenas um campo chamado *"data"* é retornado vazio, o corpo da mensagem vem com outras informações além deste campo. Mesmo assim, a Bittrex, semelhante às demais, não cumpriu a regra.

Regra 21 - 302 (*"Found"*) não deve ser utilizado.

Nenhuma *exchange* utilizou este código nos cabeçalhos da resposta.

Regra 22 - 400 (*"Bad Request"*) pode ser usado para indicar falha não específica.

A Binance, a Livecoin e a Huobi utilizam o *Bad Request*. As demais retornam uma mensagem de erro no corpo equivalente, mas com *status code* igual a 200.

Regra 23 - 401 (*"Unauthorized"*) deve ser usado quando há um problema com as credenciais do cliente.

Somente a Binance respeita a regra do código 401. A Livecoin retorna o código 400, que seria equivalente ao *"Not Found"* e as demais o código 200.

Regra 24 - 404 (*"Not Found"*) deve ser usado quando a *URI* do cliente não pôde ser mapeada.

De todas as *exchanges*, a Binance, a Livecoin e a Huobi utilizam corretamente o código 404.

Regra 25 - 405 (*"Method Not Allowed"*) dever ser utilizado quando o método *HTTP* selecionado não é suportado ou permitido.

- **Binance:** utiliza o código 404 (*"Not Found"*).
- **Bittrex:** permite que vários métodos sejam executados em um *endpoint*.
- **Poloniex:** permite que vários métodos sejam executados em um *endpoint*.
- **Livecoin:** usa corretamente o código 405.
- **Gate.io:** usa corretamente o código 405.
- **Huobi:** permite que vários métodos sejam executados em um *endpoint*.

Regra 26 - 500 (*"Internal Server Error"*) deve ser utilizado para indicar algum mal funcionamento da *API*.

- **Binance:** retorna o *status code* coerentemente, além de possuir outros códigos próprios para indicar diferentes tipos de erros internos no corpo da mensagem.
- **Bittrex:** faz uso da regra de forma correta.
- **Poloniex:** utiliza o código 200 e uma mensagem de erro própria.
- **Livecoin:** retorna uma código próprio e uma descrição no corpo da mensagem.
- **Gate.io:** retorna uma código próprio e uma descrição no corpo da mensagem.
- **Huobi:** utiliza o código 200.

3.2.4.3 Resumo

A *exchange* que mais respeitou as regras de interação com o protocolo *HTTP* foi a Binance, descumprindo apenas duas regras. A Bittrex obteve o pior desempenho descumprindo nove regras. Todas as demais descumpriram pelo menos cinco regras.

É perceptível que as corretoras ainda estão imaturas a respeito do tratamento de códigos de resposta *HTTP*. Os usuários que consumirão essas *APIs* somente em alguns casos poderão depender do *status code* para saber sobre o estado da requisição, dependendo da corretora que forem trabalhar, terão que consultar o conteúdo retornado no corpo da mensagem, podendo tornar o desenvolvimento menos simples.

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 13	Sim	Parcialmente	Sim	Sim	Sim	Sim
Regra 14	Sim	Não	Parcialmente	Parcialmente	Parcialmente	Parcialmente
Regra 15	Sim	Não	Sim	Sim	Sim	Sim
Regra 16	Sim	Não	Sim	Sim	Sim	Sim
Regra 17	Sim	Não	Não	Não	Não	Não
Regra 18	Sim	Sim	Sim	Sim	Sim	Sim
Regra 19	Sim	Sim	Não	Não	Não	Não
Regra 20	Não	Não	Não	Não	Não	Não
Regra 21	Sim	Sim	Sim	Sim	Sim	Sim
Regra 22	Sim	Não	Não	Sim	Não	Sim
Regra 23	Sim	Não	Não	Não	Não	Não
Regra 24	Sim	Não	Não	Sim	Não	Sim
Regra 25	Não	Não	Não	Sim	Sim	Não
Regra 26	Sim	Sim	Não	Não	Não	Não

Tabela 4 – Cumprimento das regras de modelagem de interação com *HTTP* por *exchange*

3.2.5 Modelagem de metadados

3.2.5.1 Cabeçalhos *HTTP*

Um conjunto de cabeçalhos padrões são definidos pelo *HTTP*, alguns deles proveem informações sobre os recursos requisitados, outros indicam informações a respeito do conteúdo que está sendo transportado na mensagem de resposta.

Regra 27 - *Content-Type* deve ser usado: este cabeçalho tem como objetivo informar o tipo de conteúdo que está sendo transmitido na resposta.

Todas as *APIs* analisadas respondem as requisições com o *Content-Type*. A exceção fica com a Huobi, que o utiliza em apenas algumas respostas.

Regra 28 - *Content-Length* deve ser ultrapassado: é o cabeçalho responsável por dizer qual o tamanho do conteúdo retornado.

A Bittrex é a única que utiliza o *Content-Length* no cabeçalho.

Regra 29 - *Last-Modified* deve ser utilizado nas repostas: informa a última vez em que o recurso alvo sofreu modificações.

Nenhuma *exchange* utilizou o campo *Last-Modified* em seus cabeçalhos, com exceção da Bittrex.

3.2.5.2 Resumo

A Bittrex foi a única corretora que teve a sua *API* respeitando as três regras supracitadas. Os consumidores que necessitam saber do tamanho do conteúdo e a data da última alteração terão que desenvolver mecanismos internos na sua aplicação para ter essas informações, podendo causar um aumento no tempo de processamento e, até mesmo, no desempenho do algoritmo.

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 27	Sim	Sim	Sim	Sim	Sim	Parcialmente
Regra 28	Não	Sim	Não	Não	Não	Não
Regra 29	Não	Sim	Não	Não	Não	Não

Tabela 5 – Cumprimento das regras de modelagem de metadados por *exchange*

3.2.6 Modelagem de representação

3.2.6.1 Representação de erros

Regra 30 - Uma forma de consistente deve ser usada para representar erros na resposta: descreve como um erro pode ser representado usando o formato *JSON*.

{ "id": valor, "descrição": texto }

- **Binance:** tem um conjunto de códigos e descrições próprios.
- **Bittrex:** retorna apenas um texto informando várias possibilidades de erro, mas não um específico.
- **Poloniex:** retorna apenas a descrição do erro.
- **Livecoin:** segundo a documentação eles possuem um padrão de representação de erro, mas na prática eles não o aplicam em todas as rotas.
- **Gate.io:** respeitam a regra e a seguem coerente.
- **Huobi:** possuem um padrão de representação, mas em erros os quais o retorno é o *status code* 404, a mensagem de erro é representada dentro de uma estrutura *HTML*.

3.2.7 Resumo

Apesar da representação de erros ser um elemento de extrema importância em uma *API*, apenas a Binance e a Gate.io cumpriram com esta regra de forma consistente.

Quando uma *API* não é capaz de informar aos seus consumidores os problemas de forma coerente e padronizada, ela dificulta a sua usabilidade, já que, dependendo do retorno que ela dá, o usuário terá mais trabalho para poder descobrir as causas daquele erro e solucionar o problema.

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 30	Sim	Não	Parcialmente	Parcialmente	Sim	Não

Tabela 6 – Cumprimento da regra de representação de erros por *exchange*

3.2.8 Preocupações do cliente

3.2.8.1 Versionamento

O versionamento é um dos principais elementos no projeto de uma *API* (DE, 2017). Elas devem sempre ser versionadas independente da abordagem seguida pela equipe de desenvolvimento. Colocar versões na sua *API* evita requisições inválidas em *URIs* atualizadas, diminui o impacto de transição entre novas versões e permite que novos detalhes sejam adicionados ou removidos causando nenhum ou pouco dano para os clientes.

Regra 31 - Novas *URIs* devem ser usadas para introduzir novos conceitos: a representação de um recurso pode mudar sua forma e/ou seu estado ao longo do tempo, porém, o identificador deste recurso na *URI* deve permanecer o mesmo caso o conceito deste recuso não se modifique. Adicionar uma identificação para a versão de um recurso - por exemplo: *v1, v2, v3* - mostra a inconsistência na conceituação deste recurso. Por via de regras, o versionamento só deve ser utilizado caso haja novos conceitos e definições a serem expostas.

- **Binance:** apenas a versão 3 está ativa e pública, significando que ela respeitou a regra.
- **Bittrex:** possui 2 versões, porém a versão mais nova está em fase de testes e a própria *exchange* não recomenda o uso.
- **Poloniex:** só possui uma versão.
- **Livecoin:** só possui uma versão.
- **Gate.io:** só possui uma versão pública.
- **Huobi:** é composta de duas versões, *market* e *v1*. Apesar de não nomear de forma padrão o indicador de versão, houve a separação de conceitos.

3.2.8.2 Segurança

Muitas *APIs* possuem informações públicas e privadas, uma aplicação de uma *exchange* de criptomoedas é um exemplo disso. Ela fornece informações que são acessíveis a todos os clientes e outras que somente usuários com conta ativa e autenticados acessam, inclusive, dados do próprio usuário.

Regra 32 - *OAuth* pode ser usado para proteger recursos: o OAuth (*Open Authorization*), ou, em tradução livre, Autorização Aberta, é um protocolo de autorização baseado no *HTTP*. Ele permite a proteção dos recursos de uma aplicação que precise de credenciais para que o seu conteúdo privado seja consumido.

Nenhuma corretora utiliza autenticação com o *Open Authorization*.

3.2.8.3 Composição da representação de resposta

O conteúdo retornado de um recurso pode ser extenso. Transmitir uma alta carga de conteúdo pela rede, tende a ser prejudicial para a largura de banda. Outro fator que implica na recepção de uma grande quantidade de informações, é o processamento que pode levar no lado do cliente. A fim de evitar os empecilhos citados, é importante que uma *API* suporte a consulta com parâmetros de limitações, sejam eles de informações do conteúdo ou de quantidade de dados.

Regra 33 - O componente de consulta de uma *URI* deve ser utilizado para dar suporte à respostas parciais: o estado de um recurso é composto de um conjunto de atributos (*fields*). O cliente deve ser capaz de consultar os recursos sem ser obrigado a receber informações indesejadas no momento. O componente de consulta (*query*) deve ser utilizado para limitar a informação e acelerar o processo de interação com a *API* por parte do usuário.

No exemplo abaixo, é feita uma consulta na *API* requisitando o nome, o cpf e a matrícula do aluno com o número de identificação (*id*) igual a 5.

https://api.ifrn.edu.br/alunos/5?fields=(nome, cpf, matricula)

Mesmo sendo um componente de grande utilidade nas *APIs*, nenhuma das seis *exchanges* oferecem suporte à essa funcionalidade.

3.2.8.4 Resumo

As *exchanges* cumpriram apenas a regra relacionada ao versionamento de *API*.

Apesar de nenhuma utilizar a autenticação com o *OAuth*, não significa que elas não sejam seguras. Cada corretora tem seus mecanismos para possibilitar o acesso às informações privadas, utilizando chaves da *API*, senhas para as chaves, informações a respeito da data e hora e criptografia.

A respeito dos componentes de consulta, nenhuma *API* os disponibiliza, podendo dificultar a manipulação de dados pelo usuário, que terá que remover ou simplesmente ignorar as informações adicionais que são retornadas.

Regras	Binance	Bittrex	Poloniex	Livecoin	Gate.io	Huobi
Regra 31	Sim	Sim	Sim	Sim	Sim	Sim
Regra 32	Não	Não	Não	Não	Não	Não
Regra 33	Não	Não	Não	Não	Não	Não

Tabela 7 – Cumprimento das regras de versionamento, segurança e composição de representação de resposta por *exchange*

3.2.9 Limites de requisições

Devido ao fato de trabalharem com transações em tempo real, as *APIs* das *exchanges* impõem restrições de uso a seus usuários. Tais limitações são referentes ao número de requisições

que podem ser efetuadas pelos clientes.

A Binance possui uma estrutura de limitações, a qual define um peso - em valor númerico inteiro - para cada rota. Quanto mais custos a operação de um *endpoin* tiver para retornar uma resposta, maior será o seu peso. Cada requisição contém um cabeçalho *Z-MBX-USED-WEIGHT* informando o peso atual utilizado para o *IP* que o executou naquele minuto.

Caso o limite de 1200 na soma dos pesos seja violado, a *API* retorná o *status code* 429 e o tempo de espera em segundos para que ela seja novamente liberada para aquele usuário. Violações repetidas após o bloqueio temporário, retornará o *status* 418 e causará um bloqueio em maior grau, que pode variar entre dois minutos e três dias.

A Livecoin utiliza um sistema de pontuações apenas para os *endpoints* da *trading API*, que são privados. É permitida apenas uma execução por segundo. Cada execução nova adiciona um ponto no sistema. Quando a pontuação ultrapassar quarenta pontos, ou seja, o limite, sua chave de acesso da *API* ficará bloqueada por oitenta segundos, com sua pontuação sendo diminuída em uma unidade a cada dois segundos, até zerar novamente. A Gate.io possui um código de erro próprio para bloqueios da *API*, porém, não especifica os limites de requisições e outras possíveis limitações.

Na Bittrex, a limitação é tratada de forma mais simples. Ela suporta sessenta requisições por minuto, com um bloqueio de um minuto caso o limite seja ultrapassado. Assim como a Bittrex, a Poloniex também é limitada à sessenta *requests* por minuto, sendo possível realizar apenas seis por segundo. Já na Huobi, é possível realizar até cem requisições em dez segundos.

4 CONSIDERAÇÕES FINAIS

Corretoras de criptomoedas existem há menos de dez anos e, devido à necessidade de consultar informações e automatizar tarefas em tempo real, algumas delas já possuem uma *API* para que seja possível acessar dados públicos e privados.

Desenvolvedores encontram dificuldade de lidar com tais *APIs* por diferentes motivos, entre eles: documentação mal escrita e mal detalhada, erros inesperados, inconsistência de dados, ausência de tratamento de erros, limitações, entre outros.

Este trabalho apresenta uma análise comparativa das *RESTful APIs* de seis diferentes *exchanges* de criptomoedas baseado em padrões e boas práticas de desenvolvimento, objetivando servir como base para outros desenvolvedores que atuam desenvolvendo soluções para o mercado de criptoativos.

A fim de atingir os objetivos da pesquisa, o desenvolvimento foi dividido em três partes: escolha das regras que serão utilizadas como parâmetros, análise individual dos padrões e do funcionamento das *APIs* conforme cada critério e levantamento das pontuações gerais obtidas pelas corretoras.

Na primeira fase, um escopo para definir as regras foi delimitado. Foram escolhidos os critérios mais básicos e necessários que todas as aplicações devem seguir, levando em consideração o contexto que essas *APIs* estão inseridas.

Na análise, o trabalho foi realizado de forma individual, para que houvesse um aprofundamento no estudo e nos testes de cada *exchange*. Além disso, foi realizada uma pesquisa a respeito das limitações de cada *API REST*, servindo, no fim, como um complemento à análise.

E, por último, um resumo de cada conjunto de regras por categoria, detalhando o cumprimento das mesmas e aprofundando na comparação e nas causas que podem levar ou não ao cumprimento de tais regras.

Durante este estudo, foi possível notar que as equipes de desenvolvimento das corretoras pouco se preocupam com as boas práticas e os padrões definidos para aplicações *RESTful*. Podendo isso ser prejudicial aos consumidores, que precisam desse serviço disponibilizado para desenvolver seus sistemas, e, até mesmo, à própria plataforma da *exchange*, com dados irreais e outros possíveis erros.

No balanço total, a Binance foi a corretora que mais cumpriu as regras, seguida da Livecoin e da Huobi. Já a Bittrex, a Gate.io e a Poloniex, cumpriram menos da metade das regras, dessa forma, mostrando que as suas *APIs* estão imaturas no quesito padronização do desenvolvimento.

4.1 LIMITAÇÕES

A principal limitação encontrada nesta pesquisa foi a falta de informações disponibilizadas pelas documentações das *APIs*. Em algumas *exchanges* analisadas, detalhes sobre os possíveis tipos de respostas de cada *endpoint*, também sobre os *status codes* e mensagens inter-

nas são omitidos. Em uma corretora específica, a Gate.io, sequer são informados os limites de requisições e o tempo de bloqueio após tal limite ser ultrapassado, dessa forma, dificultando os testes e as simulações realizadas.

4.2 TRABALHOS FUTUROS

O desenvolvimento de *APIs* passa por diversos estágios, entre eles: gerenciamento, modelagem, documentação, versionamento, testes e governança. Por ser uma área vasta e complexa, a pesquisa pode, futuramente, ser estendida analisando a documentação escrita pelas corretoras e realizando testes unitários, de performance e de segurança

O estudo das *exchanges* pode ser levado a outro patamar com o acréscimo de mais *APIs* e com a inserção de mais um objeto de análise, os *Websockets*, os quais, diferentemente das *RESTful APIs*, estão menos presentes nas corretoras.

REFERÊNCIAS

ANTONOPOULOS, A. **Mastering Bitcoin**. Califórnia: O'Reilly Media, 2014. 298 p. ISBN 14-493-7404-4.

CAETANO, R. **Learning Bitcoin**. Birmingham: Packt Publishing, 2015. 236 p. ISBN 17-852-8730-5.

DE, B. **API Management**: An architect's guide to developing and managing apis for your organization. Nova Iorque: Springer, 2017. 209 p. ISBN 14-842-1305-6.

DOGLIO, F. **REST API Development with Node.js**. 2. ed. Nova Iorque: Apress, 2018. 331 p. ISBN 14-842-3714-4.

DRESCHER, D. **Blockchain Básico: uma Introdução Não Técnica em 25 Passos**. São Paulo: Novatec, 2018. 272 p. ISBN 85-752-2669-8.

GOURLEY D. E TOTTY, B. **HTTP: The Definitive Guide: The Definitive Guide**. Califórnia: O'Reilly Media, 2002. 658 p. ISBN 15-659-2509-2.

HERNANDEZ, M. **Top 3 Online Tools for Simulating HTTP Requests**. 2017. Disponível em: <https://ubidots.com/blog/top-3-online-tools-for-simulating-http-requests/>. Acesso em: 25 mar. 2019.

LAURENCE, T. **Blockchain for Dummies**. Nova Jérsei: John Wiley and Sons, 2017. 240 p. ISBN 11-193-6559-4.

MASSÉ, M. **REST API: Design Rulebook**. 1. ed. Califórnia: O'Reilly Media, 2016. 116 p. ISBN 14-493-1050-8.

REDDY, M. **API Design for C++**. Masachusetes: Morgan Kaufmann, 2011. 446 p. ISBN 01-238-5003-7.

SCHIFF, P. **The Real Crash**: How to save yourself and your country. Nova Iorque: St. Martin's Press, 2012. 352 p. ISBN 12-500-0447-5.

SUBRAMANIAM, P. **REST API Design - Resource Modeling**. 2014. Disponível em: <https://www.thoughtworks.com/pt/insights/blog/rest-api-design-resource-modeling>. Acesso em: 28 maio 2019.

SWAN, M. **Blockchain: Blueprint for a New Economy**. Califórnia: O'Reilly Media, 2015. 152 p. ISBN 14-919-2049-7.

ULRICH, F. **Bitcoin. A Moeda na Era Digital**. São Paulo: LVM, 2014. 123 p. ISBN 85-811-9076-6.

WIKIPEDIA, A ENCICLOPEDIA LIVRE. **Cliente-servidor**. 2013. Disponível em: <https://pt.wikipedia.org/wiki/Cliente-servidor>. Acesso em: 25 mar. 2019.

WOWZA MEDIA SYSTEMS. **Peer-to-Peer Unicast Streaming. The Death of Multicast?** 2019. Disponível em: <https://www.wowza.com/resources/guides/p2p-unicast-streaming>. Acesso em: 25 mar. 2019.