UNIVERSIDADE FEDERAL DE SANTA CATARINA

Sistema de Gerência de Bancos de Dados baseado em Blockchain

Vinícius Schwinden Berkenbrock

Florianópolis
2019/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Sistema de Gerência de Bancos de Dados baseado em Blockchain

Vinícius Schwinden Berkenbrock

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.
Orientador: Prof. Jean Everson Martina, Dr.

Florianópolis
2019/2

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Berkenbrock, Vinícius Schwinden
Sistema de gerência de bancos de dados baseado em blockchain / Vinícius Schwinden Berkenbrock ; orientador, Jean Everson Martina, 2019.
146 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2019.

Inclui referências.

1. Ciências da Computação. 2. Blockchain. 3. Bancos de Dados. 4. Fragmentação de Dados. 5. Proveniência de Dados. I. Martina, Jean Everson. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Vinícius Schwinden Berkenbrock

Sistema de Gerência de Bancos de Dados baseado em Blockchain

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientador: Prof. Jean Everson Martina, Dr.

Banca Examinadora

Mestrando Lucas Machado da Palma, Bel.
Universidade Federal de Santa Catarina

Mestrando Pablo Rinco Montezano, Bel.
Universidade Federal de Santa Catarina

A todos os professores do curso, que foram tão importantes na minha vida acadêmica e no desenvolvimento deste trabalho, aos meus colegas pelo auxílio em muitas dessas mesmas etapas e à minha família que sempre me deu apoio.

AGRADECIMENTOS

Agradeço à minha mãe Karen que batalhou muito para me oferecer uma educação de qualidade e que sempre acreditou no meu potencial. Ao meu pai André, que esteve muito presente nos meus momentos de descontração. Ao meu padrasto Cid que sempre me apoiou quando precisei fazer todas as revisões textuais. Aos meus irmãos Victor, Felipe e Sarah, que me fizeram rir em tempos de puro estresse. Não posso deixar de agradecer em especial o meu orientador, Jean, que nunca negou uma ajuda durante o TCC. Por fim agradeço à todos os meus colegas de curso que contribuíram em partes com trabalhos de disciplinas que cursei, os quais inspiraram esse trabalho.

RESUMO

No cenário atual percebe-se que, os Sistemas de Gerência de Bancos de Dados Distribuídos, para realizarem escritas em diversos bancos de dados e manterem integridade, usam de técnicas de eleição de um nó responsável e este força com que os outros sejam atualizados de acordo com os pedidos aprovados pelo mesmo. Desse cenário pode decorrer os seguintes problemas: conseguindo-se adicionar clandestinamente um banco de dados no sistema ele poderia clonar os dados usados pelos demais, e após isso, quando conseguir se eleger poderia causar interrupções de escritas nos mesmos, pois poderia negar todas as requisições, por exemplo. Também poderia ocorrer a situação de queda de rede onde precisa-se eleger um novo gerente repetidamente pois ela não está consistente para manter a conexão entre os bancos, resultando em problemas de integridade em nodos que não conseguirem se comunicar com o gerente atual. O objetivo deste trabalho é implementar uma Blockchain para evitar com que os bancos de dados distribuídos possuam um ponto único de falha, denominado “gerente de bancos”, evitando problemas como queda de rede de parte dos bancos, garantir a integridade e aumentar a segurança dos dados passados entre os bancos de dados além de possibilitar o uso de diferentes tecnologias de bancos de dados. O método escolhido é o uso de uma Blockchain para garantir que haja consenso entre todos os nodos da rede, evitando assim os erros citados acima ao mesmo tempo pois não é necessário um nodo central para a gerência dos dados assim como adicionando uma camada de proveniência de dados. O resultado esperado é uma Application Programming Interface que se comunique entre dois bancos de dados estudados e uma Blockchain desenvolvida e que realize a função de “gerente de bancos”.

Palavras-chave: Blockchain. Bancos de Dados. Bancos de Dados Distribuídos. Contratos Inteligentes. Centralização. Descentralização. API. REST. Mapeador Objeto-Relacional. Sharding. Proveniência.

ABSTRACT

In the current scenario it is realized that Distributed Databases Management Systems when performing writes on multiple databases and maintain integrity, use election techniques that requires a responsible node it ensures others are updated in accordance with the requests approved by the responsible. From that scenario, the following problems can arise: by getting clandestinely a database added into the system this database could clone the data used by the others, and after that, when getting elected as the responsible could cause written interruptions in them, for it could negate all requisitions, for example. It could also occur the situation where network drops and those databases need to elect a new manager repeatedly as it is not possible to maintain the connection between the banks, resulting in problems of integrity in nodes who are unable to communicate with the current manager. The goal of the work is to implement a Blockchain to prevent distributed databases from owning a single point of failure, named "manager", avoiding problems such as falling network of part of banks, ensuring integrity and increasing the security of past data between databases beyond enabling the use of different database technologies. The chosen method is the use of a Blockchain to ensure that there is consensus among all nodes of the network, thus avoiding the errors cited above at the same time as no central nod is required for data management as well as adding a layer of data provenance. The expected result is an Application Programming Interface that communicates between two studied databases and a developed Blockchain and which performs the function of "manager".

Keywords: Blockchain. Database. Distributed Database. Smart-Contracts. Centralization. Decentralization. API. Rest. Object-Relational Mapper. Sharding. Provenance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1FN	Primeira Forma Normal
2FN	Segunda Forma Normal
3FN	Terceira Forma Normal
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	Interface(s) de Programação de Aplicações
BASE	Basicamente Disponível [Basically Available], Estado Leve [Soft state], Eventualmente consistente
BD	Banco(s) de Dados
BD NoSQL	Banco(s) de Dados Não Relacional(is)
BD SQL	Banco(s) de Dados Relacional(is)
CRUD	Criar, Ler[Read], Atualizar[Update] e Deletar
EF Core	Entity Framework Core
IP	Protocolo de Internet
IPv4	Protocolo de Internet versão 4
IPv6	Protocolo de Internet versão 6
O/RM	Mapeador Objeto-Relacional
P2P	Peer-to-Peer
REST	Transferência Representacional de Estado
SGBD	Sistema(s) de Gerência de Bancos de Dados
SGBDD	Sistema(s) de Gerência de Bancos de Dados Distribuídos
SGBD NoSQL	Sistema(s) de Gerência de Banco(s) de Dados Não Relacional(is)
SGBD SQL	Sistema(s) de Gerência de Banco(s) de Dados Relacional(is)
SOAP	Protocolo de Simples Acesso à Objetos
SQL	Linguagem(ns) de Consulta Estruturada

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo será introduzida a ideia central do trabalho, assim como uma contextualização, uma definição de problema e os objetivos esperados a serem alcançados no desenvolvimento do mesmo.

1.1 CONTEXTUALIZAÇÃO

Para contextualizar a implementação, tomaremos o cenário atual de como são armazenados os dados gerados diariamente na maior parte das empresas. Dados geralmente são apresentados geralmente de duas formas distintas: apenas Dados ou *Big Data*. A diferença entre eles é que geralmente *Big Data* muitas vezes possui um volume muito grande de dados que em sua maioria não são relevantes para quem os armazena, mas que tais dados que seriam considerados inúteis, possam ser usados em pesquisa ou para geração de novos dados a partir do que já é sabido.

Para o armazenamento desses dados hoje são empregados soluções de BD muito solidificadas no mercado, como Banco(s) de Dados Relacional(is) (BD SQL). Que em sua maioria são BD pensados para trabalhar de forma centralizada e que nem sempre são compatíveis com BD concorrentes.

Houve também uma adoção de mercado de Banco(s) de Dados Não Relacional(is) (BD NoSQL) recentemente por trabalharem melhor de forma descentralizada por não oferecer certas garantias que os BD SQL oferecem.

Contudo essas tecnologias ainda tratam a descentralização dos dados armazenados usando métodos de eleição; Esses métodos fazem com que sempre um dos BD fique responsável de transmitir dados para os demais.

Solucionando esse problema surgiram as tecnologias de *Blockchain*, que usam de redes Peer-to-Peer (P2P) para transmitir informações e armazenam dados por replicação, sendo que todos os nodos de uma rede teriam o mesmo conteúdo. Isso faz com que haja por muitas vezes, replicação excessiva de dados. Eles não usam de algoritmos de eleição de responsáveis mas trabalham de forma completamente diferente e será explicada na seção 2.2.

A poucos anos, surgiu a ideia de *Smart-Contracts* que são basicamente algoritmos de computador que rodam em nodos de uma rede *Blockchain*. Usando dessa tecnologia podemos então realizar tarefas mais complexas do que apenas armazenar dados.

Para exemplificar a implementação, tomaremos os bancos de dados usados pelo governo brasileiro, e fornecidos no Portal Brasileiro de Dados Abertos (BRASIL, 2019[a]). Eles são fornecidos em diversos formatos, muitos deles com pouca adoção de mercado, o que os torna muito difíceis de serem mantidos e de se realizar tarefas de grande valor além do que já havia sido previsto, assim limitando o seu potencial.

Temos aqui porém um problema, a base de dados brasileira é de tamanho substancial, algo que poderia tornar o uso de uma *Blockchain* inviável para o armazenamento desses dados.

Para solucionar esse problema, esse trabalho une as tecnologias de BD existentes com um controle baseado em *Blockchain*, para que não acabe ocorrendo redundância excessiva de dados.

1.2 DEFINIÇÃO DO PROBLEMA

No setor atual de tecnologia, surgiu a tecnologia chamada *Blockchain* a qual está tendo muito pouca adoção na indústria já consolidada, mas ao mesmo tempo sendo muito popular com *startups*, como apresentado por Due (2019). O motivo está justamente nas desvantagens da implantação de uma Blockchain a longo prazo: o alto consumo da capacidade física espacial dos servidores usados.

Atualmente são usados tecnologias de BD para solucionar todos os tipos de problemas relacionados à armazenamento de dados que poderiam ser solucionados em conjunto com *Blockchains*. Usam-se hoje, Sistema(s) de Gerência de Bancos de Dados (SGBD) disponíveis no mercado atual, que em sua maioria foram criados com a mentalidade de centralização, e não distribuição (como o uma *Blockchain*), de poder computacional. Alguns Sistema(s) de Gerência de Bancos de Dados Distribuídos (SGBDD) (como PostgreSQL) também foram criados com o intuito de distribuir tais arquiteturas, mas ainda são inerentemente centralizados, requisitando um nodo responsável para poder realizar suas operações (POSTGRESQL, 2019).

Outro problema está na organização da proveniência dos dados, pois os BD podem não possuir ferramentas o suficiente por si mesmos para garantir a proveniência de dados criados e/ou alterados, como dados bancários ou que possuam dados pessoais dos usuários do sistema que podem resultar em problemas graves caso estejam errados ou sejam inseridos por pessoas não autorizadas como por exemplo, emissão de passaportes que devem ser realizadas apenas por autoridades e devem ser rastreáveis, podendo assim ser explorados por usuários hackers, sendo muitas vezes necessário a reversão do BD para o último estado válido, e nem sempre com garantia total de que esse estado escolhido não havia sido previamente atacado.

1.3 OBJETIVOS

Nessa seção são apresentados os objetivos gerais e específicos.

1.3.1 Objetivo Geral

Para solucionar os problemas apresentados anteriormente, de centralização de processamento de dados e proveniência dos mesmos além de muitos outros, esse

trabalho propõe justamente a união dessas duas tecnologias, *Blockchain* e SGBD, que trabalham de formas opostas (*Blockchain* sendo descentralizada e SGBD sendo centralizados), para criar uma plataforma única de gerência de BD.

Será apresentado nesse trabalho, uma solução para o problema de armazenamento brasileiro, que será representado por dados de censos escolares, de modo a evitar redundância desnecessária de dados, ao mesmo tempo que seja solucionado o problema de validação dos dados, de forma automática. Além disso, o sistema irá permitir o uso de diversas tecnologias distintas, de forma que possa ser implantado com os dados já existentes e dados futuros.

1.3.2 Objetivos Específicos

Nesse trabalho irão ser desenvolvidos:

- dois SGBD que usem tecnologias diferentes, conhecidos por *MongoDB* e *SQLite*.
- um *Smart-Contract* baseado na rede *Ethereum* para controle de meta-dados.
- uma API de comunicação com a *Blockchain Ethereum*.
- uma API de comunicação com o *MongoDB*.
- uma API de comunicação com o *SQLite*.
- uma API de controle que irá realizar comunicações entre as API citadas acima e o usuário.

Para a população de itens dentro dos BD serão usados dois *datasets* disponíveis na seção de educação que provêm dados sobre as instituições de ensino básico (BRASIL, 2019[b]) e as médias dos alunos por turma (BRASIL, 2019[c]).

Será realizada a engenharia reversa de ditas planilhas, usando as técnicas de formas normais até a terceira forma normal (ver 2.1.2) e desenvolvido o gráfico de entidades relacionais, assim como a divisão de qual BD será responsável por quais dados, realizando, desta forma, um *Sharding* (ver 2.1.5.1) dos dados existentes.

1.4 ESTRUTURA DO TRABALHO

Este trabalho encontra-se estruturado da seguinte maneira:

- o capítulo 2 detalha as tecnologias usadas, como BD, *Blockchain*, API e redes de forma geral.
- o capítulo 3 apresenta um apanhado geral do que é mais atual nesta área de pesquisa com os detalhes sobre levantamento de dados utilizados.

- o capítulo 4 contém detalhes da proposta, mais especificamente sobre a modelagem e implementação do sistema proposto, assim como os passos realizados para a reprodução de resultados.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta a fundamentação teórica sobre a área de estudo deste trabalho. Serão tratadas as áreas de BD, Blockchain e Redes no contexto da criação de um sistema distribuído.

2.1 BANCOS DE DADOS

O termo BD, surgiu na década de 60 juntamente com o surgimento das tecnologias de discos rígidos segundo o dicionário Oxford (2013). Com o surgimento desses componentes, houve a possibilidade de acessar os dados de forma não sequencial, o que permitia que houvesse a necessidade de um sistema de hierarquia e localização desses dados.

Foi então que na década de 70, com a publicação do artigo de Codd (1970), surgiram os SGBD, um conjunto de aplicação de gerência de um ou mais BD que permitiam a abstração do controle dos BD para um sistema único.

Antes do surgimento desses SGBD, eram feitos programas de gerência que tinham módulos específicos para realizar as operações Criar, Ler[Read], Atualizar[Update] e Deletar (CRUD). Além disso havia a necessidade de saber a localização exata dos dados e quais dados foram criados, o que acarretava em muitos problemas para os programadores da época. Os bancos de dados surgiram para suprir essa demanda.

Usando a afirmação "um banco de dados nada mais é do que um sistema de armazenamento de dados baseado em um computador"(DATE, 2004, p. 26). Partindo dessa afirmação e de outros dados contidos no seu livro, podemos também especificar mais os BD em diferentes tipos, sendo eles os de hierarquia, os relacionais e os pós-relacionais.

O(s) BD SQL são os BD mais usados no mercado segundo (DB-ENGINES, 2019). Eles ocupam cerca de 75% do mercado atual de bancos de dados e desde sua criação tiveram uma adoção muito forte no mercado.

Bancos de dados são geralmente representados na forma de tabelas, o que facilita a sua visualização. Sendo que cada tabela seria referente a um tipo de dado, as colunas dessa tabela são chamadas de atributos e cada linha corresponde a um dado do tipo da tabela. Pode-se visualizar um exemplo na Figura 1.

2.1.1 Bancos de Dados Relacionais

Os BD SQL são bancos que são descritos por tabelas onde cada linha dessa tabela representa um dado. Essas tabelas são ligadas entre si por meios de relacionamentos e por isso são chamados de relacionais.

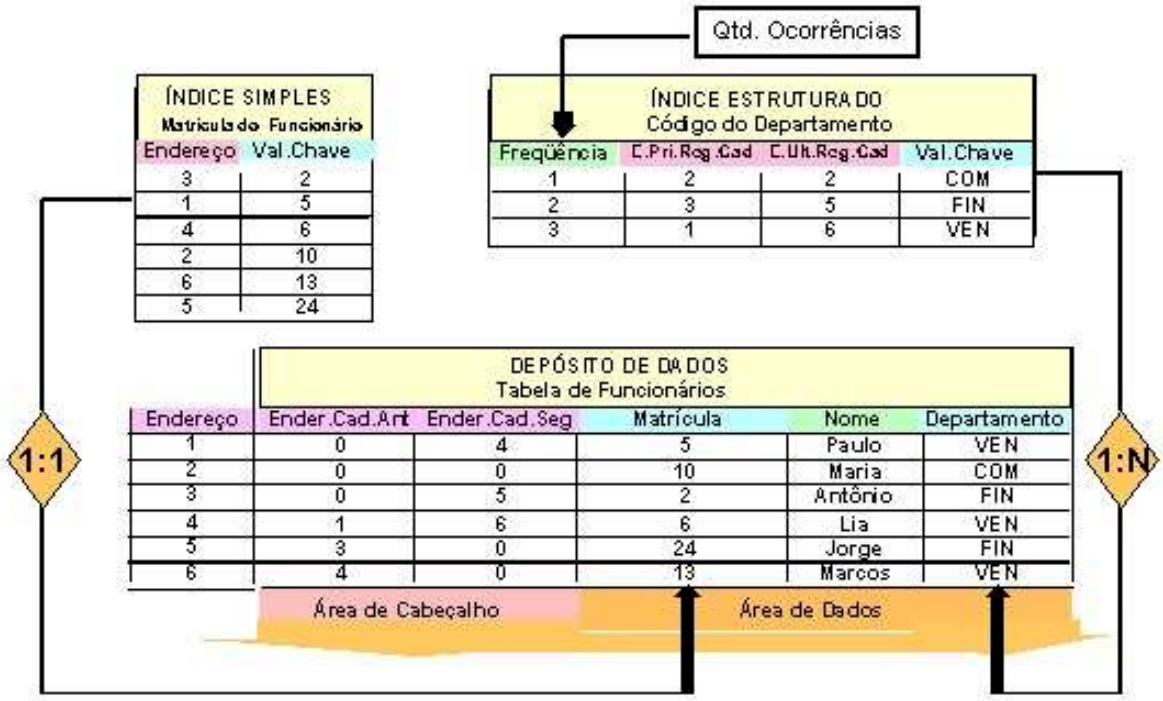


Figura 1 – Exemplo de Esquema de BD

A grande diferença dos BD SQL para os bancos que existiam na época que foi inventado é a possibilidade de realizar operações algébricas nas pesquisas ao banco, fazendo com que eles recebessem muito mais velocidade de processamento dos dados.

A maioria dos BD SQL possuem uma linguagem de consulta padronizada chamada de Linguagem(ns) de Consulta Estruturada (SQL). E os Sistema(s) de Gerência de Banco(s) de Dados Relacional(is) (SGBD SQL) geralmente possuem como padrão o sistema Atomicidade, Consistência, Isolamento e Durabilidade (ACID) como base das transações que o SGBD realiza.

2.1.1.1 ACID

Segundo (WIKIPEDIA, 2019), o sistema ACID foi inventado em 1983 e amplamente incorporado nos BD SQL para reduzir conflitos quando há mais de um acesso ao BD e ele garante que:

- ao menos as transações só serão realizadas caso elas possam ser realizadas por completo ou irão falhar e o BD não será modificado.
- a transação só poderá ser realizada caso ela traga o BD para um estado válido.
- um conjunto de transações realizadas paralelamente só poderá ser realizado, caso o resultado final seja idêntico ao resultado desse mesmo conjunto sendo

realizado de forma sequencial.

- uma vez que a transação foi realizada, ele deverá ser mantida mesmo em caso de falha do sistema.

2.1.2 Formas Normais

Os conceitos dessa seção e de suas subseções foram baseados no trabalho de Date (2004).

Com o desenvolvimento dos BD houve a necessidade de organizar os dados para que se evitasse repetição dos dados e ao mesmo tempo, que impedisse uma omissão de relação entre dados relacionados, tendo-se assim uma melhor forma de representar um dado em um banco de dados.

A técnica de normalização foi inventada para solucionar esse caso, e para que houvesse uma modularização de como ela devesse ser aplicada houve uma divisão em 6 formas normais distintas principais, as quais somente serão apresentadas as 3 primeiras Formas Normais, por se tratarem das mais usadas e proverem um bom desempenho, relativo à uma forma não normal, para o BD. (DATE, 2004, p. 223).

2.1.2.0.1 Primeira Forma Normal

A Primeira Forma Normal (1FN) requer que todos os atributos de um dado sejam únicos, ou seja, não sejam compostos por uma lista de elementos. Ela também reforça que todas as tabelas possuam uma Chave Primária que é um atributo do dado onde esse não pode ser repetido na tabela.

Um exemplo seria uma tabela com o "número de telefones"dos "clientes"de uma loja. Caso algum cliente tenha mais de um telefone, o correto seria criar uma tabela extra onde essa teria uma relação de clientes e telefones e a tabela antiga ter apenas os dados do cliente.

2.1.2.0.2 Segunda Forma Normal

A Segunda Forma Normal (2FN) requer primeiramente que o BD já esteja na 1FN. Partindo do pressuposto anterior, ela afirma que qualquer atributo ou conjunto de atributos que não depende de outros atributos ou conjunto de atributos deve ser separado da tabela e criado uma nova tabela.

Um exemplo seria ter uma extensão da tabela anterior contendo um Fabricante, um Produto, o Modelo do Produto e o país do Fabricante. Nesse caso, dividiria-se a tabela em uma contendo apenas os dados do Fabricante e do seu País, e outra contendo os dados do Produto, ou seja, o Fabricante, o Produto e o Modelo do Produto.

2.1.2.0.3 Terceira Forma Normal

A Terceira Forma Normal (3FN) requer primeiramente que o BD já esteja na 2FN. Partindo do pressuposto anterior, ela afirma que qualquer atributo ou conjunto de atributos que não depende da chave primária da tabela, deve ser separado em outra tabela.

Um exemplo seria uma continuação da tabela apresentada anteriormente, onde o Modelo do Produto teria informações de quem criou esse modelo, quem aprovou esse modelo e assim por diante.

Para que essa tabela fosse compreendida na 3FN, teria de separar os dados do Modelo do Produto da tabela Produto, ou seja, a primeira tabela teria o Produto, que contém o Fabricante, o Produto e uma referência para o Modelo do Produto, e a segunda tabela conteria o Modelo do Produto, o Criador do Modelo e o Aprovador do Modelo.

2.1.3 Bancos de Dados Não Relacionais

Segundo Leavitt (2010), os BD NoSQL surgiram da necessidade de processamento de dados modelados de forma a não necessariamente ter relacionamentos entre tabelas. Esses BDs existiram desde a década de 60 mas não possuíam a designação "NoSQL"até o início do século 21.

Eles tiveram um aumento de adoção devido ao crescimento horizontal (ou seja, adição de processamento com novas máquinas ligadas por rede) dos sistemas usados hoje em dia. Eles conferem um melhor desempenho em sistemas distribuídos de forma clusterizada que é um problema para os BD SQL que são bons para crescimento vertical (ou seja, adição de poder de processamento ao núcleo existente).

Os Sistema(s) de Gerência de Banco(s) de Dados Não Relacional(is) (SGBD NoSQL) possuem diversas formas e representações de dados mas a maioria deles adere ao sistema Basicamente Disponível [Basically Available], Estado Leve [Soft state], Eventualmente consistente (BASE) de funcionamento como padrão - porém alguns também suportam o sistema ACID - fazendo com que eles sejam muito mais simples e flexíveis.

2.1.3.1 BASE

O sistema BASE foi inventado para que os SGBD NoSQL tenham um desempenho mais rápido ao executar transações por relaxarem um pouco os requisitos impostos por um sistema ACID. A base desse tipo de sistema consiste em:

- o sistema irá realizar leituras e escritas da maneira mais rápida possível, mesmo que isso comprometa na consistência dos dados.

- o sistema não tendo garantia de consistência deve, depois de algum tempo, ter uma probabilidade de convergir a um estado consistente.
- se o sistema está funcionando e esperarmos tempo o suficiente, deveremos eventualmente ter o sistema em um estado consistente.

2.1.4 Mapeador Objeto-Relacional

Um Mapeador Objeto-Relacional (O/RM) é um sistema de abstração usado em linguagens orientadas a objetos, para que os objetos sejam convertidos de forma automática para um BD SQL. Essa abstração faz com que o desenvolvimento seja feito de forma mais familiar aos desenvolvedores, sem a necessidade de conhecer a SQL.

Um objeto é composto de uma classe e seus atributos, esses são mapeados para modelos (tabelas) e cada objeto criado é adicionado nesse modelo como sendo uma linha. Os atributos dessas classes são mapeados como colunas.

2.1.5 Bancos de Dados Distribuídos

Segundo (ÖZSU; VALDURIEZ, 2011), o(s) SGBDD são SGBD que usam de BD distribuídos em uma rede de computadores. Esses SGBDD diferentemente dos SGBD tradicionais, são sistemas que realizam as transações de forma distribuída, de forma que caso um dado X esteja distribuído em dois BD diferentes, X1 e X2, a pesquisa seja realizada de forma independente em X1 e X2 e depois sejam agregados em um nodo principal.

Existem algumas formas de implementação desses sistemas, sendo as mais atuais replicação e duplicação. Replicação envolve o desenvolvimento de rotinas de alta complexidade e de alto consumo computacional e com desempenho inferior ao de duplicação. A duplicação por sua vez, escolhe um BD como sendo o principal, e todos os outros BD são cópias desse BD principal.

Esses SGBDD também podem ser classificados em dois tipos, homogêneos ou heterogêneos. Os SGBDD homogêneos trazem a característica de suportar apenas um tipo de BD, enquanto os heterogêneos suportam mais de um tipo. Os sistemas heterogêneos geralmente também precisam de um sistema de comunicação feita geralmente usando a SQL.

Os diferentes BD em um SGBDD também podem ser projetados usando duas técnicas distintas, por replicação ou por fragmentação (ou particionamento), por replicação, um BD tem os mesmos dados que outro BD, e por fragmentação um BD teria dados diferentes de outros BD. Geralmente essas técnicas são usadas em conjunto.

2.1.5.1 Sharding

A técnica de *Sharding*, constitui na distribuição horizontal e vertical de um BD, onde o SGBDD precisa passar por todos os nodos que ele possui, de modo que consiga reconstituir todos os dados armazenados, pois certos dados podem estar sendo enviados apenas para certos SGBD.

2.2 BLOCKCHAIN

Uma forma de implementação para um sistema de pagamentos conhecido como Bitcoin foi proposto por NAKAMOTO (2009), onde existe um maior detalhamento sobre as necessidades desse sistema que ficou conhecido posteriormente como Blockchain.

Uma Blockchain é basicamente um conjunto de transações (2.2.2) válidas agregadas em blocos (2.2.2) onde esses blocos são imutáveis. Portanto uma Blockchain é levemente diferente de um BD pois ao invés de adicionar, atualizar ou remover dados, ela armazena as transações que foram realizadas e para alterar o estado de uma certa transação é armazenado outra transação ao invés de apenas atualizar a existente. Além disso, todos os nodos de uma rede *Blockchain* possuem uma cópia completa de tudo que foi armazenado, consumindo consideravelmente mais espaço que um BD normal que pode conter partes dos dados armazenados em diferentes locais.

Contudo esse "BD"deve ser distribuído usando uma rede P2P para todos os usuários desse sistema, o que causa uma replicação massiva de dados e algo gasto de poder computacional para realizar validações de transações (no modelo proposto por NAKAMOTO (2009)).

Uma Blockchain também pode implementar um sistema de estados chamados Smart-Contracts (2.2.6), onde é armazenado o estado de como um contrato e as transações que ocorreram para que ele seja alterado.

2.2.1 Redes e Peer-to-Peer

As redes necessárias para a Blockchain geralmente são completamente P2P, ou seja, um nodo (usuário) da internet comunica-se com outros diretamente, não necessitando passar por um servidor principal. Isso faz com que esse tipo de rede tenha algumas vantagens e desvantagens em relação ao cliente-servidor que é mais amplamente conhecido.

O modelo da Internet foi criado pensando no uso principal do modelo cliente-servidor, onde os servidores teriam uma largura de banda maior. Isso acarreta na desvantagem do cliente tendo menos largura de banda para realizar envios dos dados, sendo assim mais lento que se um servidor estivesse enviando os dados. A vantagem é a sua disponibilidade, que por não depender de um servidor, pode realizar transações desde que mais de um usuário esteja conectado à essa rede.

No caso da Blockchain, os Blocos 2.2.2 são considerados válidos quando são aceitos por mais de 50% dos nodos Mineiradores (2.2.3) da rede.

2.2.1.1 Protocolo IPv4

O Protocolo de Internet versão 4 (IPv4) é a quarta versão do Protocolo de Internet (IP). Ele é o mais utilizado atualmente, mesmo que esteja sendo continuamente substituído pelo Protocolo de Internet versão 6 (IPv6).

O IP fornece para a Internet como um todo, um sistema de endereçamento, onde podemos localizar alguma máquina em específico se tivermos conhecimento do seu código IP. Quando usamos o IPv4 temos suportes para algumas formas de endereçamento onde podemos realizar múltiplas cópias de um envio de dados para muitos nodos simultaneamente, sem precisar recorrer a técnicas de laços de repetição, que acabariam fazendo com que os repetidores da internet (roteadores) ficassem sobrecarregados.

Essas técnicas são chamados modos de endereçamento e são definidas a seguir:

- Unicast: Esse é o método mais comum, onde um nodo envia para apenas mais um nodo o dado. Esse modo de endereçamento é obrigatório ter suporte no IPv4.
- Broadcasting: Permite que todos os nodos em um determinado espaço de IP recebam os dados enviados de uma só vez. Esse modo de endereçamento é obrigatório ter suporte no IPv4.
- Multicasting: Permite que todos os nodos que estiverem registrados nos roteadores recebam os dados de uma só vez. Os nodos devem então enviar um sinal para os roteadores dizendo que querem receber aqueles dados. Esse modo de endereçamento é opcional ter suporte no IPv4.
- Anycast: Permite que um nodo envie um dado para o nodo mais próximo. Esse modo de endereçamento é obrigatório ter suporte no IPv4.

2.2.1.2 Protocolo IPv6

O IPv6 é a sexta versão do IP. Ele é o protocolo escolhido para substituir o protocolo IPv4.

A diferença do IPv6 para o IPv4 é que ele possui mais espaços de endereçamento e não fornece suporte ao Broadcast, fazendo com que caso seja necessário, seja usado o endereçamento Multicast em alguns endereços específicos, sendo agora obrigado a ter suporte no IPv6.

2.2.2 Blocos e Transações

Os blocos em uma Blockchain são definidos como um registro de todas as transações realizadas de forma que ele não possa ser mudado com o tempo. Esses blocos também devem ser ligados entre si, para que haja uma ordem cronológica entre eles e para que não possa ocorrer de ser incluídos blocos não desejados no meio da Blockchain.

Transações são registros de alterações de dados que são realizadas por nodos. Essas transações precisam ser identificadas pelos nodos para que possa haver validação pelos Mineradores (2.2.3).

Usa-se de algoritmos *hash* para que seja garantido a integridade desses blocos, impossibilitando a falsificação dos blocos e das transações. Ver 2.2.4.

2.2.3 Minerador

O trabalho de um minerador é receber transações da maioria (quando possível, de todos) os nodos da rede, e conferir se essa transação pode ocorrer. Para um registro de transações simples, isso é algo trivial, mas para um registro de gastos de dinheiro por exemplo (que é onde é mais usado atualmente (2.2.5)), ele deve percorrer todos os blocos para verificar se a transação pode ser efetuada ou não.

Posteriormente ele deve pegar as transações que são válidas, processá-las para que não possam ser alteradas usando algoritmos de criptografia, agrupá-las em um Bloco, ligar o Bloco atual com o Bloco anterior, e por fim processar o bloco no mesmo algoritmo para que ele não possa ser falsificado na rede.Após criado o Bloco, ele deve distribuir na rede esse Bloco.

Para que seja feito todo esse processamento, geralmente é dado ao minerador um incentivo na forma de moedas daquela plataforma.

2.2.4 Função Hash

Uma função *hash* é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Uma função *hash* criptográfica permite verificar facilmente alguns mapeamentos de dados de entrada para um valor *hash* fornecido, mas se os dados de entrada são desconhecidos, é deliberadamente difícil reconstruí-lo (ou alternativas equivalentes) conhecendo o valor do *hash* armazenado. Isto é usado para assegurar a integridade de dados.

2.2.5 Bitcoin e Altcoins

O Bitcoin foi inicialmente proposto em um artigo por (NAKAMOTO, 2009) onde ele cria a especificação e a prova de conceito. É um sistema de criptomoedas onde ocorrem registros de transações monetárias da moeda chamada Bitcoin.

Com o passar do tempo, surgiram diversas moedas novas conhecidas atualmente como Altcoins. Geralmente elas diferem do Bitcoin por proverem algum serviço ou suporte a mais funcionalidades que o mesmo.

A Altcoin mais famosa atualmente é chamada de Ethereum. Essa Altcoin possui diversas características diferentes, mas será mais ressaltado a sua capacidade de execução de Contratos Inteligentes (2.2.6).

O trabalho irá realizar o uso da moeda Ethereum e sua plataforma de Contratos Inteligentes para realizar parte das funcionalidades propostas.

2.2.6 Contratos Inteligentes

Contratos Inteligentes (ou Smart-Contracts), são algoritmos de computador que permitem a realização de um contrato entre duas ou mais entidades sem que elas precisem de uma entidade central como um banco ou cartório para que ocorra esse contrato.

Ele possui diversos mecanismos de segurança para que esse contrato seja realmente efetivado, mas que para o escopo deste trabalho não precisa ser entendido.

2.2.7 Proveniência de Dados

Esse é um modo de uso muito utilizado atualmente para garantir que a proveniência de um dado é válida. Geralmente ela é armazenada em algum banco seguro. Muitas vezes é usado Blockchains para garantir tais proveniências são imutáveis quando aceitas.

2.3 INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES

Segundo BRAUNSTEIN (2018) "em termos não técnicos, uma API é um contrato que diz para os desenvolvedores de software que onde sempre que um cliente enviar uma requisisão para um servidor em um formato específico, você irá sempre receber uma resposta no mesmo formato ou iniciar uma ação definida."

Essa é uma definição meio ampla, e por isso será comparado na próxima subseção, os modelos de API mais usadas no mercado atual.

2.3.1 SOAP e REST

Uma API pode ter vários formatos e modelos diferentes de ser feito, entre eles os mais usados na indústria são os formatos Transferência Representacional de Estado (REST) e Protocolo de Simples Acesso à Objetos (SOAP). Ambos formatos são muito poderosos, mas trazem abordagens completamente diferentes.

Uma diferenciação feita entre essas duas abordagens foi feita por (SOAPUI.ORG, 2019) na qual foram comparadas as vantagens de cada tipo de formato.

Segue abaixo os principais pontos positivos de se usar o formato SOAP:

- Não depende de tecnologias de transporte sobre IP como *HTML* que é requerido pelo formato REST.
- Funciona bem para setores empresariais altamente distribuídos que quando comparado ao REST funciona de modo ponto-a-ponto.
- É padronizado.
- Pode ser extendido com o padrão *WS*.
- Possui tratamento de erros.
- Possui recursos de automatização quando usado com certas linguagens.

Segue abaixo os principais pontos positivos de se usar o formato REST:

- Possui padrões fáceis de entender.
- Menor curva de aprendizado.
- Rápido pois não possui processamento de informações.
- Próximo de outras tecnologias usadas atualmente para desenvolvimento *Web*.

3 ESTADO DA ARTE

O presente capítulo apresenta o levantamento do estado da arte e como se encontram as pesquisas relacionadas a Blockchain. Essa revisão bibliográfica seguiu a metodologia de revisão sistemática de literatura definida por MARCONI e LAKATOS (2003).

3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO

O objetivo da revisão sistemática da literatura neste trabalho é levantar os fatos e dados relevantes e atuais sobre o uso de Blockchains como coordenador de tarefas, podendo assim descobrir falhas e contextos não cobertos dentro desse caso de uso. Com o pensamento de cumprir os objetivos deste levantamento do estado da arte, um protocolo de busca foi criado e está descrito abaixo.

3.1.1 Base de Dados e Strings de Busca

Na tabela 2 são listados os termos de busca que podem ser utilizados nos principais sites de busca por artigos (IEEE, ACM Digital Library, Google Scholar).

Tabela 2 – Termos Usados nas Pesquisas.

Termos	Sinônimos	Tradução (Inglês)
Blockchain	-	
Contrato Inteligente	-	Smart Contract
Coordenador	Administrador	Manager/Coordinator
Banco de Dados	-	Database
Proveniência de Dados	-	Data Provenance

Na tabela 3 são listados os repositórios de artigos científicos que foi realizado a pesquisa, em conjunto das *strings* de busca utilizadas nas respectivas bases de dados. Vale salientar que as *strings* aplicadas em cada repositório são equivalentes. Vale ressaltar que essas strings usaram os termos definidos na tabela 2.

3.1.2 Critérios de Inclusão/Exclusão

Após o processo de elaboração das strings de busca para cada uma das fontes, um número alto de trabalhos foram encontrados. Com o intuito de filtrar apenas os trabalhos relevantes à pesquisa, alguns critérios de inclusão e exclusão foram definidos.

Critérios de inclusão:

- Apenas trabalhos escritos nas línguas inglesa e portuguesa.

Tabela 3 – Strings de Busca.

Repositório	Campo de Busca
IEEE Xplore Digital Library	(("Blockchain"OR "Smart Contract") AND (("Database"AND ("Manager"OR "Coordinator")) OR "Data Provenance"))
Google Scholar	((("Blockchain"OR "Smart Contract") AND (("Database"AND ("Manager"OR "Coordinator")) OR "Data Provenance")) OR (("Blockchain"OR "Contratos Inteligentes") AND (("Banco de Dados"AND ("Administrador"OR "Coordenador")) OR "Proveniência de Dados")))
ACM Digital Library	(("Blockchain"OR "Smart Contract") AND (("Database"AND ("Manager"OR "Coordinator")) OR "Data Provenance"))

- Citar soluções que usam Blockchain como apoio de um BD.
- Citar o uso de Blockchain como coordenador de tarefas.

Critérios de exclusão:

- O uso exclusivo de um *Blockchain* para armazenamento de dados.

3.2 EXECUÇÃO DA BUSCA

Esta busca por trabalhos na área de pesquisa foi realizada em dois momentos, nos dias 10/02/2019 e 20/10/2019, para isso foram utilizadas as strings de busca presentes na tabela 3. A tabela 4 exibe a quantidade de artigos encontrados durante a revisão do estado da arte. Vale salientar que foram feitos refinamentos e testes com as *strings* de buscas para que este resultado fosse obtido.

Como é possível de se perceber, a tabela é dividida em 3 etapas:

1. Busca Inicial: o número de artigos presentes nesta coluna corresponde ao número total de artigos encontrados.
2. Filtro 1: o número de artigos presentes nesta coluna foram os que tiveram títulos e resumos lidos e tiveram alguma relevância com o tema desse projeto.
3. Filtro 2: exibe o número de artigos que foram lidos por completo, selecionados da etapa anterior.

3.3 EXTRAÇÃO DE INFORMAÇÃO E ANÁLISE DOS RESULTADOS

À partir da busca, apresentada na seção anterior 3.2, foram encontrados 110 artigos ao todo, sendo que apenas 8 foram de extrema importância e passaram o

Tabela 4 – Numero de artigos encontrados durante revisão literária.

Repositório	Busca Inicial	Filtro 1	Filtro 2
IEEE Xplore Digital Library	21	5	3
Google Scholar	30	0	0
ACM Digital Library	59	8	5
Total	110	13	8

segundo filtro apresentado na tabela 4. Como esta é uma área relativamente nova na computação em geral, houve dificuldade na busca de artigos que abordassem a temática "*Blockchain* como coordenador de tarefas", sendo tarefa qualquer assunto que possa ser coordenado, como procedimentos de algoritmos distribuídos ou transações de bancos de dados, principalmente porque as aplicações e sistemas existentes focam apenas na parte de armazenamento da proveniência dos dados na *Blockchain*.

3.4 DISCUSSÃO DOS RESULTADOS

Tabela 5 – Comparação entre artigos e assuntos abordados.

Artigos	Blockchain	Proveniência de Dados	Blockchain guarda meta-dados de um BD	Blockchain Coordena um BD	Sharding
(KELLER; KESSLER, 2018)	x	x			
(LIANG *et al.*, 2017)	x	x	x		
(CUI *et al.*, 2019)	x	x	x		
(RAMACHANDRAN; RAMACHANDRAN, 2018)	x	x	x		
(EZHILCHELVAN; ALDWEESH; MOORSEL, 2018)	x		x	x	
(DANG *et al.*, 2019)	x				x
(BRAGAGNOLO *et al.*, 2018)	x				
(BRAGAGNOLO *et al.*, 2019)	x				
Esta Proposta	x	x	x	x	x

Detalhando a tabela acima, a coluna *Blockchain* deve-se o uso de uma *Blockchain* ou *Smart-Contract* na implementação da solução provida pelo artigo; a coluna *Proveniência de Dados* implica que foi explorado o uso da *Blockchain* para garantir um sistema de Proveniência de Dados; a coluna *Blockchain guarda meta-dados de um BD* implica no uso de uma *Blockchain* para guardar meta-dados de um BD ao invés de guardar os dados em si economizando assim espaço na *Blockchain* em si; a coluna *Blockchain Coordena um BD* implica que a solução utilizada faz com que a *Blockchain* coordene um sistema de BD; a coluna *Sharding* implica no uso de técnicas de *Sharding* ou exploração dessas técnicas (ver em 2.1.5.1) para a solução desenvolvida.

É possível verificar na tabela 5 que os trabalhos encontrados não focam em todos os aspectos relevantes do escopo dessa proposta na sua maioria por não forçar

com que o BD usado para armazenamento de dados tenha o seu acesso depois de ter um comando da *Blockchain*.

Como pode ser observado, mesmo os trabalhos mais completos focados em mostrar um sistema em Blockchain que guarda meta-dados de um BD acabam focando mais na proveniência dos dados do que no controle e *sharding* do BD.

4 PROPOSTA

Nesse capítulo tem-se como objetivo detalhar a proposta central desse projeto. Mais especificamente o como pode ser criado um conjunto de API para inter-comunicação de sistemas distintos e independentes, usando um módulo em *Blockchain* para controle e metadados, um módulo para comunicação e controle entre as API e módulos de consulta de BD distintos.

Para que isso ocorra, será necessário resolver as etapas abaixo:

- Modelagem e Implementação de uma API de inter-comunicação. Essa API serve para o usuário do sistema fazer requisições para o sistema, realizando as operações internas necessárias para enviar uma resposta para o mesmo usuário quando necessário.
- Modelagem e Implementação de um *Smart-Contract* e uma API para comunicação com o dito *Smart-Contract*. O *Smart-Contract* deverá manter um controle de quais dados estão gravados em quais BD usando metadados dos dados fornecidos previamente pelo administrador do sistema.
- Modelagem e Implementação dois BD e suas respectivas API. Estes tem a função de armazenar os dados fornecidos pelo usuário através da API de inter-comunicação e responder à consultas da mesma API.

As etapas de modelagem acima serão detalhadas nas seções 4.2, 4.3 e 4.4 e de implementação nas seções 4.9, 4.6, 4.8 e 4.7.

Para realizar a modelagem e implementação dos BD, é necessário também possuir um cenário de aplicação para que o mesmo possa ser implementado e testado. Para fins de reprodutibilidade, o cenário escolhido usa um conjunto de dados fornecido pelo governo brasileiro de forma pública (BRASIL, 2019[a]).

A linguagem escolhida para implementação do sistema como um todo foi C# por conveniência do autor.

4.1 EXEMPLO DE CENÁRIO DE APLICAÇÃO

O cenário de aplicação escolhido foi o de educação básica brasileira. Para isso usamos de dois conjuntos de dados disponíveis em (BRASIL, 2019[b]) e (BRASIL, 2019[c]), usando dos dados mais recentes disponíveis.

Nesse cenário temos que os órgãos governamentais e empresas privadas precisam saber de informações sobre censos e estatísticas de suas escolas para que possam realizar medidas apropriadas na melhora dos seus serviços.

4.2 MODELAGEM DAS INTERFACES DE PROGRAMAÇÃO DE APLICAÇÃO

O paradigma das API utilizado é o paradigma REST. As API serão divididas em 3 categorias como descrito acima, uma API de comunicação, onde irá realizar as operações que o usuário enviar e irá coordenar as demais API; uma API para comunicação com a *Blockchain*, que se tornará responsável de armazenar e consultar metadados armazenados no *Smart-Contract*; e duas API de comunicação com dois diferentes BD.

A Figura 2 apresenta a modelagem conceitual da intercomunicação das API desenvolvidas. Vale lembrar que a API usada é do tipo REST.

4.3 MODELAGEM DO *SMART-CONTRACT*

A *Blockchain* escolhida para a realização deste trabalho é a *Ethereum*, a qual fornece suporte à *Smart-Contract* ao mesmo tempo sendo a *Blockchain* mais usada para esse fim.

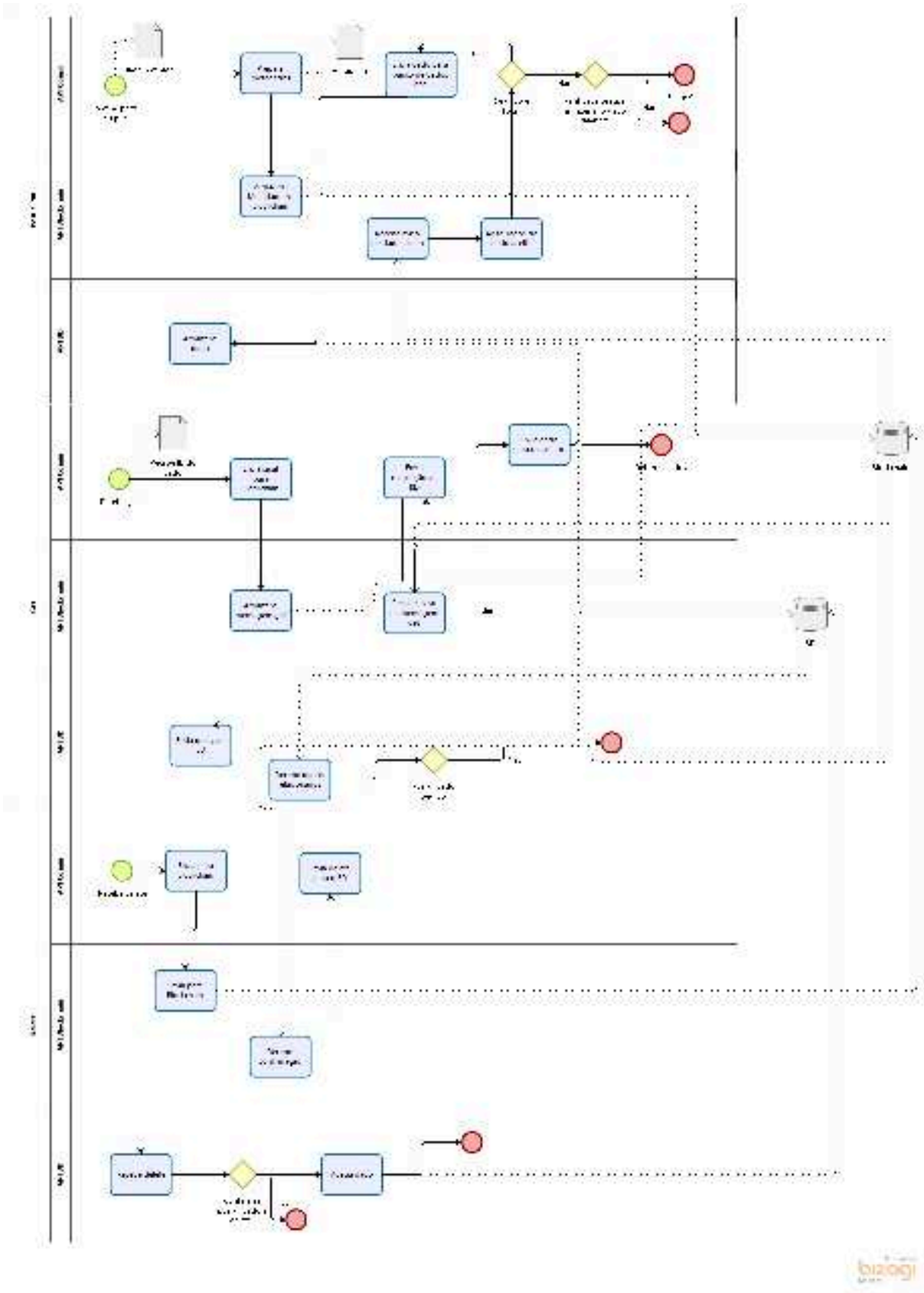
O *Smart-Contract* foi feito usando a linguagem Solidity, que é a mais usada, e a com maior suporte da comunidade, tendo em vista que as demais ou possuem uma falta de suporte ou estão depreciadas.

O *Smart-Contract* deverá ter como meta-dados:

- Meta-dados dos Dados Global:
  - Hash do Dado.
  - Status do Dado (se foi deletado ou não).
- Meta-dados das Tabelas:
  - Mapeamento dos meta-dados dos dados que da Tabela.
  - Lista dos meta-dados dos atributos dos dados da Tabela.
  - Numero de dados de uma Tabela.
- Meta-dados dos BD:
  - Meta-dados de quais tabelas estão no BD.
  - Meta-dados de quais dados estão na tabela desse BD. (*Sharding* vertical).
  - Meta-dados de quais atributos dos dados estão na tabela. (*Sharding* horizontal).

Além disso o *Smart-Contract* deverá possuir eventos para alertar a API própria quando ocorrer alguma mudança para que ela avise a API de controle quais BD estão envolvidos nessa mudança e caso algum deles precise realizar ações. Esses eventos são:

Figura 2 – Modelagem do funcionamento das APIs.



- Evento de Criação de BD.
- Evento de Criação de Tabela com Codificação de Atributos e adição da mesma no BD. Isso faz com que haja possibilidade de replicação dos dados quando necessário e permite *Sharding*.
- Evento de Criação de Dado Globalmente.
- Evento de Adição de Dado inteiro ou parcial em uma Tabela.
- Evento de Modificação de Dado Globalmente.
- Evento de Deleção de Dado Globalmente.

No caso dos *Smart-Contracts* sempre que há um evento, também é registrado quem realizou aquele evento em um Log. Esse log pode ser acessado para realizar estudos de proveniência.

4.4 MODELAGEM DOS BANCOS DE DADOS

As tecnologias para o desenvolvimento de BD, foram escolhidas pela familiaridade com suas ferramentas e boas características para uso no cenário. Segue abaixo os SGBD escolhidos:

- BD SQL: PostgreSQL.
- BD NoSQL: MongoDB.

Foi criado também em nível de API, uma abstração no conceito de BD, usando um O/RM conhecido por Entity Framework Core (EF Core).

O esquema do BD foi primeiramente modelado, usando a 1FN, nos dados retirados do (BRASIL, 2019[a]). Originalmente os dados fornecidos já estavam nesse formato. Em seguida foi aplicado a 2FN, a qual pouco mudou o esquema. Termina-se a modelagem usando a 3FN que mudou radicalmente o esquema. O Apêndice A apresenta o resultado final da 3FN.

Com a 3FN, é possível gerar o Diagrama Entidade Relacional apresentado na Figura 3 e com base nela determinar quais atributos que ocorrem de estar ausentes em alguns dados, esses dados serão armazenados no BD NoSQL. Os atributos que estão sempre presentes, serão adicionados no BD SQL.

Além disso foi decidido por retirar as tabelas "Censo"e "Dependência Administrativa"e adicionar seus atributos na tabela "Censo_Escola"por gerarem complicações desnecessárias ao BD.

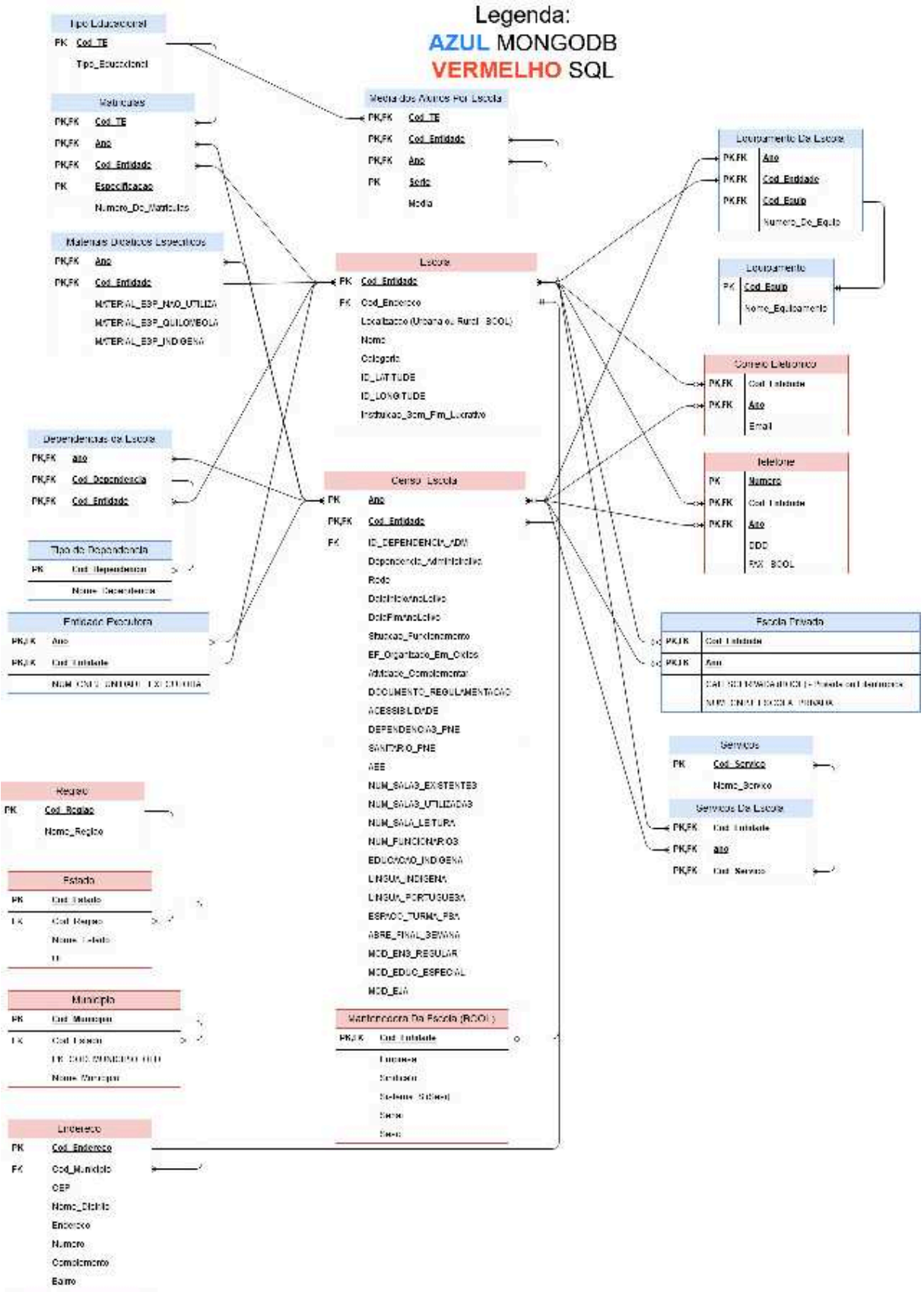


Figura 3 – Diagrama Entidade Relacional

4.5 DESENVOLVIMENTO DO SMART-CONTRACT

O *Smart-Contract* pode ser encontrado no Apêndice M.

Primeiramente começamos com as estruturas que irão guardar os meta-dados do contrato. Iniciamos com a estrutura de atributos, onde ela irá guardar as identidades dos atributos, como um enumerador, ao mesmo tempo que guarda quais tabelas possuem tais atributos. Em seguida a estrutura para guardar os meta-dados dos dados, onde ela irá ficar responsável pelos Ids dos dados para que esses Ids sejam únicos assim como as tabelas que possuem esse dado. Por conseguinte, a estrutura para guardar os meta-dados das tabelas também guarda quais dados estão em cada tabela assim como aponta para os BD que possuem essa tabela, além disso, irá guardar os atributos que essa tabela possui. Por fim, a estrutura dos BD onde guarda-se o endereço IP do BD assim como quais tabelas estão registradas no mesmo.

```
1  contract BlockchainToBD {
2      struct Attributes {
3          uint256 attsId;
4          uint64 numTables;
5          mapping(uint64 => Table) tables;
6      }
7
8      struct Data {
9          bytes32 hash;
10         bool deleted;
11
12         uint64 numTables;
13         mapping(uint64 => Table) tables;
14     }
15
16     struct Table {
17         uint128 numData;
18         mapping(uint128 => Data) dataInTable;
19
20         uint256 numAtt;
21         mapping(uint256 => Attributes) atts;
22
23         uint32 numDB;
24         mapping(uint32 => DB) dbs;
25     }
26
27
28     struct DB {
29         uint64 numTables;
30         mapping(uint64 => Table) tablesInDB;
31
32         bytes32 ipAdd;
```

```
33          bytes32 typeOfDB;
34      }
```

Em seguida precisamos de variáveis para armazenar os BD e os atributos, tendo em vista que as operações serão realizadas a partir dos mesmos. Precisamos também de uma variável que irá armazenar o código do BD para questões de segurança.

Para garantir a integridade dos dados, foi preciso criar uma função que irá conferir se o *Hash* apresentado para conferência é o mesmo do dado a ser conferido. (ver 2.2.4).

Para popular o *Smart-Contract* é preciso criar funções de adição de BD, Tabelas, Atributos e Dados.

Para completar as funções, precisa-se adicionar as funções de manipulação de dados, de forma que sejam emitidos eventos para que os BD sejam notificados para que se atualizem.

A idéia dessa implementação é que os eventos relatem para os BD quais tabelas e dados precisam ser atualizados. A abordagem poderia ter sido diferente, mas alguns meta-dados precisariam ser guardados em cada BD, o que pode não ser desejado. Essa mudança de abordagem iria diminuir custos na plataforma *Ethereum* uma vez que não seria necessário um *loop* triplo.

Por fim temos a implementação dos eventos, para que os BD sejam notificados de alterações no contrato que influenciam a si mesmos. Alguns desses eventos são necessários pois não existe como pegar os retornos das funções do contrato através da API *Nethereum*.

Lembrando que sempre é guardado o *sender*, garantindo assim a proveniência dos dados.

4.6 DESENVOLVIMENTO DA API DE BLOCKCHAIN

Essa seção foi separada em duas subseções para abordar os Controladores e o Início da API de forma separada.

4.6.1 Inicio da API

Primeiramente, todo programa em C# começa com o arquivo Program.cs, que cria o *Host* do programa nesse caso, e chama as inicializações do programa na classe Startup.cs.

Por consequência o Startup.cs é invocado e aqui inicializamos o contrato e os controladores, além do sistema de *Pooling* de eventos. Veja em R.

O parâmetro IConfiguration é configurado no appsettings.json como informado abaixo. Esse serve para poder padronizar alguns valores globais para esse projeto, sem colocá-los explicitamente no código.

```
{
        "Logging": {
                "LogLevel": {
                        "Default": "Information",
                        "Microsoft": "Warning",
                        "Microsoft.Hosting.Lifetime": "Information"
                }
        },
        "Ethereum": {
                "url": "http://localhost:8545",
                "private":
                ↪  "0xb5b1870957d373ef0eeffecc6e4812c0fd08f554b37b233526acc331bf15
                "contractAddress": ""
        },
                "AllowedHosts": "*"
        }
```

Por fim, a classe EventPool serve para receber quando ocorreram alterações nos dados e enviar pedidos para a API de comunicação. Veja em Q.

4.6.2 Controladores

O Construtor da API de controle de Atributos recebe o serviço usado pela *Blockchain*, para que esse serviço seja acessado a garanta a conexão com o *Smart-Contract*. A Implementação do Controlador de Atributos deve pensar em adicionar atributos e conectar um atributo a uma tabela, foi usado Post para a Criação e Put para a conexão. Ver no Apêndice S.

O Construtor da API de controle de meta-dados de BD recebe o serviço usado pela *Blockchain*, para que esse serviço seja acessado a garanta a conexão com o *Smart-Contract*. A implementação desse controlador deve pensar em resgatar o IP de um BD assim como adicionar novas entradas. Ver no Apêndice T.

O Construtor da API de controle de meta-dados de dados recebe o serviço usado pela *Blockchain*, para que esse serviço seja acessado a garanta a conexão com o *Smart-Contract*. A implementação desse controlador deve pensar em várias funções, sendo elas em ordem: verificação dos dados, adicionar dados, atualizar dados, deletar dado de uma tabela, deletar dado de todas as tabelas com os atributos daquele dado. Ver no Apêndice U.

O Construtor da API de controle de meta-dados de tabelas recebe o serviço usado pela *Blockchain*, para que esse serviço seja acessado a garanta a conexão com o *Smart-Contract*. A implementação deve pensar apenas na adição de tabelas em

certos BD. Ver no Apêndice V.

4.7 DESENVOLVIMENTO DA API DE BANCO DE DADOS NOSQL

Primeiramente foi criado um projeto usando o *Template* de criação de API usando .Net Core.

Logo após a criação por Template foi desenvolvido a entidade que seria usada para comunicar com o banco de dados. Ver no Apêndice Y.

Em seguida foi desenvolvido o contexto no qual o BD NoSQL será acessado. Ver no Apêndice X.

Para a definição das configurações do BD também foi implementado uma interface e classe de auxílio. Ver no Apêndice W.

Usando essa classe de auxílio, foi alterado o appsettings.json para informar os parâmetros usados no BD Mongo.

```
{
        "MongoDBSettings": {
                "ExtrasCollectionName": "Extras",
                "ConnectionString": "mongodb://localhost:27017",
                "DatabaseName": "ExtrasDb"
        },
        "Logging": {
                "LogLevel": {
                        "Default": "Information",
                        "Microsoft": "Warning",
                        "Microsoft.Hosting.Lifetime": "Information"
                }
        },
        "AllowedHosts": "*"
}
```

Definidas as propriedades básicas do BD, podemos então programar a API em si. Como todo programa escrito na linguagem C#, usa-se a convenção de iniciar um programa usando o arquivo Program.cs para realizar sua inicialização.

Uma API desenvolvida em C# precisa também ser inicializada, e no arquivo Startup.cs é onde ela é realizada, com um atento para a introdução de serviços *Singletons*, essa é a configuração recomendada pelos desenvolvedores do MongoDB.

```
9  namespace API.Mongo
10 {
11         public class Startup
12         {
13                 public Startup(IConfiguration configuration)
14                 {
15                         Configuration = configuration;
16                 }
17
18                 public IConfiguration Configuration { get; }
19
20                 // This method gets called by the runtime. Use this method to add services to the container.
21                 public void ConfigureServices(IServiceCollection services)
22                 {
23                         services.Configure<MongoDBSettings>(
24                                 Configuration.GetSection(nameof(MongoDBSettings)));
25
26                         services.AddSingleton<IMongoDBSettings>(sp =>
27                                 sp.GetRequiredService<IOptions<MongoDBSettings>>().Value);
28
29                         services.AddSingleton<ExtrasDaEscolaContext>();
30
31                         services.AddControllers();
32                 }
33
```

```
34              // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
35              public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
36              {
37                      if (env.IsDevelopment())
38                      {
39                              app.UseDeveloperExceptionPage();
40                      }
41
42                      app.UseHttpsRedirection();
43
44                      app.UseRouting();
45
46                      app.UseAuthorization();
47
48                      app.UseEndpoints(endpoints =>
49                      {
50                              endpoints.MapControllers();
51                      });
52              }
53      }
54  }
```

Por fim a definição da API em si é feita através de um controlador. Esse controlador segue a convenção e chama-se MongoController.cs.

Leva-se em consideração que como o *MongoDB* é um BD de documentos, todos os dados de uma certa Entidade e Ano de censo (Definidos por *Indexer* na implementação) serão armazenados no mesmo local. Caso o usuário precise apenas de parte dos dados, apenas seria necessário fazer uma classe com os atributos a serem buscados no BD e o *Driver* (pacote disponibilizado pela *Microsoft*, de uso padrão nas implementações de *MongoDB*) do *MongoDB* irá automaticamente formatar a saída com os dados corretos.

Essa implementação foi feita para que seja completamente independente e não possui nada de diferente de uma implementação usada no mercado atual.

4.8 DESENVOLVIMENTO DA API DE BANCO DE DADOS SQL

Primeiramente foi criado um projeto usando o *Template* de criação de API usando .Net Core.

No Apêndice B encontra-se o *script* de criação do banco de dados. A partir dele, foi criado o BD e com isso realizamos um procedimento chamado *scaffold* que o EF Core suporta. Com ele é realizado o mapeamento do BD para entidades (ver Apêndices D a L) e contexto (ver Apêndice C) diretamente para o BD escolhido. O comando foi descrito abaixo.

```
dotnet ef dbcontext scaffold
↪ "Host=localhost;Database=postgres;Username=postgres;Password=postgres"
↪ Npgsql.EntityFrameworkCore.PostgreSQL -o Models
```

Ele realiza a conexão ao BD em localhost, usando o BD postgres com usuário e senha postgres que são padrões de instalação. Ele usa o provedor Npgsql para realizar as tarefas descritas acima e envia os arquivos gerados para a pasta Models dentro do projeto.

Logo após, foi alterado o arquivo *appsettings.json* para adicionar as propriedades de conexão.

```
{
        "Logging": {
                "LogLevel": {
                        "Default": "Information",
                        "Microsoft": "Warning",
                        "Microsoft.Hosting.Lifetime": "Information"
                }
        },
```

```
9         "AllowedHosts": "*",
10        "ConnectionStrings": {
11                "escolasContext":
                  ↪  "Host=localhost;Database=postgres;Username=postgres;Password=po
12        }
13  }
```

Em seguida foram movidas as propriedades de configuração da entidade gerada como *postgresContext.cs* para o *Startup.cs*.

Em seguida geramos os controladores de API usando o gerador de controlador de API usando Entity Framework padrão e alteramos os controladores, pela necessidade dos mesmos de usar chaves compostas.:

- *CensoEscola* (Ver Apêndice N)
- *Telefone* (Ver Apêndice O)
- *CorreioEletronico* (Ver Apêndice P)

Os demais controladores foram mantidos como padrão, não estando aqui referenciados por falta de relevância, para evitar poluição do conteúdo.

Por fim, foi adicionado um algoritmo, no Program.cs, para garantir que o BD existe e caso contrário, que ele seja criado para agilizar no *deploy*.

```
8   namespace API.SQL
9   {
10          public class Program
11          {
12                  public static void Main(string[] args)
13                  {
14                          var host = CreateHostBuilder(args).Build();
15
16                          CreateDbIfNotExists(host);
17
18                          host.Run();
19                  }
20
21                  private static void CreateDbIfNotExists(IHost host)
22                  {
23                          using (var scope = host.Services.CreateScope())
24                          {
25                                  var services = scope.ServiceProvider;
```

```
26
27                                  try
28                                  {
29                                          var context =
                                            ↪  services.GetRequiredService<postgresCon
30                                          context.Database.EnsureCreated();
31                                  }
32                                  catch (Exception ex)
33                                  {
34                                          var logger =
                                            ↪  services.GetRequiredService<ILogger<Pro
35                                          logger.LogError(ex, "An error
                                            ↪  occurred creating the DB.");
36                                  }
37                          }
38                  }
39                  public static IHostBuilder CreateHostBuilder(string[]
                    ↪  args) =>
40                          Host.CreateDefaultBuilder(args)
41                                  .ConfigureWebHostDefaults(webBuilder =>
42                                  {
43                                          webBuilder.UseStartup<Startup>();
44                                  });
45          }
46
47
48  }
```

4.9 DESENVOLVIMENTO DA API DE COMUNICAÇÃO

Primeiramente, todo programa em C# começa com o arquivo Program.cs, que cria o *Host* do programa nesse caso, e chama as inicializações do programa.

Por consequência o Startup.cs é invocado e aqui inicializamos os controladores.

Também possui uma classe *Handler* de eventos para o *Pooling* do *Blockchain* (4.6). Essa classe é a responsável por realizar as atualizações de BD. Ver no Apêndice A.

Foi criado também uma classe auxiliar para o *Handler* e o *Pooling* de teor Global. Ver no Apêndice B.

Os controladores usados aqui na verdade apenas consomem as API criadas previamente nos outros sistemas. Abaixo tem um exemplo de como é o funcionamento

geral. Esse esquema repete-se para todas as API desenvolvidas. A idéia é que o usuário final do sistema comunique-se apenas por essa API. Um detalhe importante é que os construtores podem receber uma uri diferente da padrão, para que possa ser usado mais de um banco (com API independente) usando o mesmo controlador. Ver no Apêndice C.

4.10 CONTRIBUIÇÕES

As principais contribuições desse trabalho são:

- Implementação e documentação de um sistema modular de integração de diferentes BD através de uma *Blockchain*, o qual pode ser adicionado em qualquer sistema existente com pouca ou nenhuma alteração necessária no sistema de produção.
- Possibilidade de realização de *Sharding* e proveniência de dados de maneira simples e altamente visível.
- Possibilidade de abstração do uso de uma Blockchain.
- Nova forma de realizar gerência de BD de forma distribuída.
- Novo paradigma de controle usando API.

4.11 METODOLOGIA DE PESQUISA

Este trabalho seguirá o seguinte fluxo:

1. Levantamento da fundamentação teórica.
2. Levantamento do Estado da Arte e práticas do mercado atual.
3. Especificação do modelo de BD a ser usado.
4. Especificação do modelo de Blockchain a ser usado.
5. Engenharia reversa do cenário de aplicação para desenvolvimento dos BD.
6. Implementação dos BD do cenário de aplicação escolhido.
7. Implementação de uma API para comunicação para cada BD.
8. Implementação de um *Smart-Contract* para ser usado em uma Blockchain.
9. Implementação de uma API para comunicação com a Blockchain.
10. Implementação de uma API para inter-comunicação de API.

11. Execução de testes.
12. Escrita da Monografia.

4.12 CONCLUSÃO

Esse trabalho conseguiu implementar uma arquitetura para ambientes que precisam garantir a veracidade e proveniência dos dados e que não se importam com os problemas de atraso acarretados da *Blockchain*, ao mesmo tempo que, realiza um uso eficiente da *Blockchain* ao armazenar apenas meta-dados dos dados armazenados nos BD.

4.13 TRABALHOS FUTUROS

A seguir é provido uma lista com alguns temas de trabalhos futuros sugeridos:

- Substituição do Contrato Inteligente por uma *Blockchain* própria a fim de aumentar o desempenho e diminuir o tamanho da *Blockchain* em si e talvez aumentar a generalização proposta para que tenha mais propósitos.
- Ampliação do uso, criando novos cenários de aplicação.
- Criação de bibliotecas genéricas para rápido desenvolvimento de novos sistemas.
- Ao invés de a *Blockchain* coordenar diferentes BD, coordenar diferentes nodos em uma rede de processamento paralelo, assim diminuindo a dependência de um nodo mestre como ocorre atualmente na área de processamento paralelo de dados.
- Aplicar conhecimentos para utilização em nuvem e contêineres.

REFERÊNCIAS

BRAGAGNOLO, Santiago *et al.* BlockchainDB - Towards a Shared Database on Blockchains. Association for Computing Machinery, Amsterdã, jul. 2019.

BRAGAGNOLO, Santiago *et al.* Ethereum Query Language. Association for Computing Machinery, Gotemburgo, maio 2018.

BRASIL, República Federativa do. **Conjuntos de Dados - Portal Brasileiro de Dados Abertos.** Disponível em: `http://dados.gov.br/dataset`. Acesso em: 23 set. 2019.

BRASIL, República Federativa do. **Instituições de Ensino Básico - Conjuntos de Dados - Portal Brasileiro de Dados Abertos.** Disponível em: `http://dados.gov.br/dataset/instituicoes-de-ensino-basico`. Acesso em: 23 set. 2019.

BRASIL, República Federativa do. **Instituições de Ensino Básico - Conjuntos de Dados - Portal Brasileiro de Dados Abertos.** Disponível em: `http://dados.gov.br/dataset/media-de-alunos-por-turma-na-educacao-basica`. Acesso em: 23 set. 2019.

BRAUNSTEIN, Mark L. **Health Informatics on FHIR: How HL7's New API is Transforming Healthcare.** 1. ed. Atlanta: Springer, 2018.

CODD, Edgar Frank. A Relational Model of Data for Large Shared Data Banks. Association for Computing Machinery, California, 1970.

CUI, Hongyan *et al.* IoT Data Management and Lineage Traceability: A Blockchain-based Solution. Institute of Electrical e Electronics Engineers, China, set. 2019.

DANG, Hung *et al.* Towards Scaling Blockchain Systems via Sharding. Association for Computing Machinery, Amsterdã, 2019.

DATE, C. J. **Introdução a Sistema de Bancos de Dados, Tradução da 8a edição Americana.** 8. ed. Rio de Janeiro: Elsevier, 2004.

DUE. **Blockchain Adoption Barriers in Startups and Enterprises.** Ago. 2019. Disponível em: `https://due.com/blog/blockchain-adoption-barriers-in-startups-and-enterprises/`. Acesso em: 23 set. 2019.

DB-ENGINES. **DB-Engines Ranking per database model category**. Disponível em: `https://db-engines.com/en/ranking_categories`. Acesso em: 19 out. 2019.

EZHILCHELVAN, P.; ALDWEESH, A.; MOORSEL, A v. Non-blocking two-phase commit using blockchain. Association for Computing Machinery, Nova Iorque, jun. 2018.

KELLER, T.; KESSLER, N. Yet Another Blockchain Use Case – The Label Chain. Institute of Electrical e Electronics Engineers, China, dez. 2018.

LEAVITT, Neal. Will NoSQL Databases Live Up to Their Promise? Institute of Electrical e Electronics Engineers Computer Society, Estados Unidos, fev. 2010.

LIANG, X. *et al.* ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. Institute of Electrical e Electronics Engineers, Espanha, maio 2017.

MARCONI, M. de A.; LAKATOS, E. M. **Fundamentos de metodologia científica.** 5. ed. São Paulo: Atlas, 2003.

NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System., 2009. Disponível em: `https://bitcoin.org/bitcoin.pdf`. Acesso em: 11 nov. 2018.

OXFORD. **OED Online.** Oxford: Oxford University Press, 2013. Disponível em: `http://www.oed.com/view/Entry/47411`.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems.** 3. ed. Nova Iorque: Springer-Verlag, 2011.

POSTGRESQL. **Documentation: 12: Chapter 26, High Availability, Load Balancing and Replication**. Disponível em: `https://www.postgresql.org/docs/12/high-availability.html`. Acesso em: 23 set. 2019.

RAMACHANDRAN, A.; RAMACHANDRAN, A. SmartProvenance: A Distributed, Blockchain Based DataProvenance System. Association for Computing Machinery, Nova Iorque, mar. 2018.

SOAPUI.ORG. **SOAP vs REST 101: Understand The Differences.** Disponível em: `https://www.soapui.org/learn/api/soap-vs-rest-api.html`. Acesso em: 23 set. 2019.

WIKIPEDIA. **Sistemas de gerênciamentos de bancos de dados.** Disponível em: `https://en.wikipedia.org/wiki/ACID`. Acesso em: 23 set. 2019.

Apêndices

APÊNDICE A – TABELA DE TERCEIRA FORMA NORMAL

Tabela 6 – Terceira Forma Normal

Censo	Ano													
escola	COD_ENTIDADE	Nome	Localizacao (Urbana ou Rural - BOOL)	Categoria	ID_LATITUDE	ID_LONGITUDE	Instituicao_Sem_Fim_Lucrativo	Cod_Endereco						
Censo-Escola	Ano	Cod_Entidade	DataInicioAnoLetivo	DataFimAnoLetivo	Situacao_Funcionamento	EF_Organizado_Em_Ciclos	Atividade_Complementar	DOCUMENTO_REGULAMENTACAO	ACESSIBILIDADE	DEPENDENCIAS_PNE	SANITARIO_PNE	AEE		
Censo-Escola (Continuação)	NUM_SALAS_EXISTENTES	NUM_SALAS_UTILIZADAS	NUM_SALA_LEITURA	NUM_FUNCIONARIOS	EDUCACAO_INDIGENA	LINGUA_INDIGENA	LINGUA_PORTUGUESA	ESPACO_TURMA_PBA	ABRE_FINAL_SEMANA	MOD_ENS_REGULAR	MOD_EDUC_ESPECIAL	MOD_EJA	ID_DEPENDENCIA_ADM	
Dependência Administrativa	ID_DEPENDENCIA_ADM	Dependencia_Administrativa	Rede											
Regiao	Cod_Regiao	Nome_Regiao												
Estado	Cod_Estado	Nome_Estado	Cod_Regiao	UF										
Municipio	Cod_Municipio	PK_COD_MUNICIPIO_OLD	Nome_Municipio	COD_ESTADO										
Endereco	Cod_Endereco	CEP	Cod_Municipio	Nome_Distrito	Endereco	Numero	Complemento	Bairro						
Correio Eletronico	Ano	Cod_Entidade	Email											
Telefone	Ano	Cod_Entidade	DDD	Numero	Fax (BOOL)									
Escola Privada	Ano	Cod_Entidade	CATESCPRIVADA (BOOL) - Privada ou Filantropica	NUM_CNPJ_ESCOLA_PRIVADA										
Entidade Executora	Ano	Cod_Entidade	NUM_CNPJ_UNIDADE_EXECUTORA											
Mantenedora Da Escola(BOOL)	Cod_Entidade	Empresa	Sindicato	Sistema_S (Sesi)	Senai	Sesc								
Dependencias da Escola	Cod_Entidade	Cod_Dependencia	Ano											
Tipo de Dependencia	Cod_Dependencia	Nome_Dependencia												
Serviços (Alim, Agua, Lixo, Energia)	Cod_Servico	Nome_Servico												
Servicos Da Escola	Cod_Entidade	Cod_Servico	Ano											
Equipamento	Cod_Equip	Nome_Equipamento												
Equipamento da Escola	Cod_Entidade	Cod_Equip	Numero_De_Equip	Ano										
Materiais Didaticos Especificos	Cod_Entidade	MATERIAL_ESP_NAO_UTILIZA	MATERIAL_ESP_QUILOMBOLA	MATERIAL_ESP_INDIGENA	Ano									
Matriculas	Cod_Entidade	Cod_TE	Especificacao	Numero_De_Matriculas	Ano									
Tipo Educacional	Cod_TE	Tipo_Educacional												
Media de alunos Escolas	Cod_Entidade	Cod_TE	Serie	Media	Ano									

APÊNDICE B – CRIAÇÃO DE TABELAS SQL

Abaixo está o script de criação do BD SQL.

```
-- Table: public.regiao

-- DROP TABLE public.regiao;

CREATE TABLE public.regiao
(
    cod_regiao bigint NOT NULL,
    nome_regiao text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT regiao_pkey PRIMARY KEY (cod_regiao)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.regiao
    OWNER to postgres;

-- Table: public.estado

-- DROP TABLE public.estado;

CREATE TABLE public.estado
(
    cod_estado bigint NOT NULL,
    "UF" character(2) COLLATE pg_catalog."default" NOT NULL,
    nome_estado text COLLATE pg_catalog."default" NOT NULL,
    cod_regiao bigint NOT NULL,
    CONSTRAINT estado_pkey PRIMARY KEY (cod_estado),
    CONSTRAINT estado_cod_regiao_fkey FOREIGN KEY (cod_regiao)
        REFERENCES public.regiao (cod_regiao) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
WITH (
```

```
37      OIDS = FALSE
38  )
39  TABLESPACE pg_default;
40
41  ALTER TABLE public.estado
42      OWNER to postgres;
43
44
45  -- Table: public.municipio
46
47  -- DROP TABLE public.municipio;
48
49  CREATE TABLE public.municipio
50  (
51      cod_municipio bigint NOT NULL,
52      cod_estado bigint NOT NULL,
53      pk_cod_municipio_old bigint NOT NULL,
54      nome_municipio text COLLATE pg_catalog."default" NOT NULL,
55      CONSTRAINT municipio_pkey PRIMARY KEY (cod_municipio),
56      CONSTRAINT municipio_cod_estado_fkey FOREIGN KEY (cod_estado)
57          REFERENCES public.estado (cod_estado) MATCH SIMPLE
58          ON UPDATE NO ACTION
59          ON DELETE NO ACTION
60          NOT VALID
61  )
62  WITH (
63      OIDS = FALSE
64  )
65  TABLESPACE pg_default;
66
67  ALTER TABLE public.municipio
68      OWNER to postgres;
69
70  -- Table: public.endereco
71
72  -- DROP TABLE public.endereco;
73
74  CREATE TABLE public.endereco
75  (
```

```
76      cod_endereco bigint NOT NULL,
77      cod_municipio bigint NOT NULL,
78      cep text COLLATE pg_catalog."default" NOT NULL,
79      nome_destrito text COLLATE pg_catalog."default" NOT NULL,
80      endereco text COLLATE pg_catalog."default" NOT NULL,
81      numero text COLLATE pg_catalog."default" NOT NULL,
82      complemento text COLLATE pg_catalog."default" NOT NULL,
83      bairro text COLLATE pg_catalog."default" NOT NULL,
84      CONSTRAINT endereco_pkey PRIMARY KEY (cod_endereco),
85      CONSTRAINT endereco_cod_municipio_fkey FOREIGN KEY (cod_municipio)
86          REFERENCES public.municipio (cod_municipio) MATCH SIMPLE
87          ON UPDATE NO ACTION
88          ON DELETE NO ACTION
89          NOT VALID
90  )
91  WITH (
92      OIDS = FALSE
93  )
94  TABLESPACE pg_default;
95
96  ALTER TABLE public.endereco
97      OWNER to postgres;
98
99  -- Table: public.escola
100
101 -- DROP TABLE public.escola;
102
103 CREATE TABLE public.escola
104 (
105     cod_entidade bigint NOT NULL,
106     cod_endereco bigint NOT NULL,
107     localizacao boolean,
108     nome text COLLATE pg_catalog."default" NOT NULL,
109     categoria text COLLATE pg_catalog."default",
110     id_longitude text COLLATE pg_catalog."default",
111     id_latitude text COLLATE pg_catalog."default",
112     instituicao_sem_fim_lucrativo text COLLATE pg_catalog."default",
113     CONSTRAINT escola_pkey PRIMARY KEY (cod_entidade),
114     CONSTRAINT escola_cod_endereco_fkey FOREIGN KEY (cod_endereco)
```

```
115         REFERENCES public.endereco (cod_endereco) MATCH SIMPLE
116         ON UPDATE NO ACTION
117         ON DELETE NO ACTION
118         NOT VALID
119 )
120 WITH (
121     OIDS = FALSE
122 )
123 TABLESPACE pg_default;
124
125 ALTER TABLE public.escola
126     OWNER to postgres;
127
128 -- Table: public.mantenedora_da_escola
129
130 -- DROP TABLE public.mantenedora_da_escola;
131
132 CREATE TABLE public.mantenedora_da_escola
133 (
134     cod_entidade bigint NOT NULL,
135     empresa boolean NOT NULL,
136     sindicato boolean NOT NULL,
137     sistems_s_sesi boolean NOT NULL,
138     senai boolean NOT NULL,
139     sesc boolean NOT NULL,
140     CONSTRAINT mantenedora_da_escola_pkey PRIMARY KEY (cod_entidade),
141     CONSTRAINT mantenedora_da_escola_cod_entidade_fkey FOREIGN KEY
        ↪ (cod_entidade)
142         REFERENCES public.escola (cod_entidade) MATCH SIMPLE
143         ON UPDATE NO ACTION
144         ON DELETE NO ACTION
145         NOT VALID
146 )
147 WITH (
148     OIDS = FALSE
149 )
150 TABLESPACE pg_default;
151
152 ALTER TABLE public.mantenedora_da_escola
```

```
153     OWNER to postgres;
154
155 -- Table: public.censo_escola
156
157 -- DROP TABLE public.censo_escola;
158
159 CREATE TABLE public.censo_escola
160 (
161     id_dependencia_adm smallint,
162     dependencia_administrativa text COLLATE pg_catalog."default",
163     rede text COLLATE pg_catalog."default",
164     data_inicio_ano_letivo date,
165     data_fim_ano_letivo date,
166     situacao_funcionamento text COLLATE pg_catalog."default",
167     ef_organizado_em_ciclos boolean,
168     atividade_complementar text COLLATE pg_catalog."default",
169     documento_regulamentacao text COLLATE pg_catalog."default",
170     acessibilidade boolean,
171     dependencias_pne text COLLATE pg_catalog."default",
172     sanitarios_pne text COLLATE pg_catalog."default",
173     aee text COLLATE pg_catalog."default",
174     num_salas_existentes bigint,
175     num_salas_usadas bigint,
176     num_salas_leitura bigint,
177     num_funcionarios bigint,
178     educacao_indigena boolean,
179     lingua_indigena boolean,
180     lingua_portuguesa boolean,
181     espaco_turma_pba boolean,
182     abre_final_semana boolean,
183     mod_ens_regular boolean,
184     mod_educ_especial boolean,
185     mod_eja boolean,
186     ano smallint NOT NULL,
187     cod_entidade bigint NOT NULL,
188     CONSTRAINT censo_escola_pkey PRIMARY KEY (ano, cod_entidade),
189     CONSTRAINT censo_escola_cod_entidade_fkey FOREIGN KEY (cod_entidade)
190         REFERENCES public.escola (cod_entidade) MATCH SIMPLE
191         ON UPDATE NO ACTION
```

```
192         ON DELETE NO ACTION
193         NOT VALID
194 )
195 WITH (
196     OIDS = FALSE
197 )
198 TABLESPACE pg_default;
199
200 ALTER TABLE public.censo_escola
201     OWNER to postgres;
202
203 -- Table: public.correio_eletronico
204
205 -- DROP TABLE public.correio_eletronico;
206
207 CREATE TABLE public.correio_eletronico
208 (
209     cod_entidade bigint NOT NULL,
210     ano smallint NOT NULL,
211     email text COLLATE pg_catalog."default" NOT NULL,
212     CONSTRAINT correio_eletronico_pkey PRIMARY KEY (cod_entidade, ano),
213     CONSTRAINT correio_eletronico_cod_entidade_fkey FOREIGN KEY
      ↪ (cod_entidade)
214         REFERENCES public.escola (cod_entidade) MATCH SIMPLE
215         ON UPDATE NO ACTION
216         ON DELETE NO ACTION
217         NOT VALID
218 )
219 WITH (
220     OIDS = FALSE
221 )
222 TABLESPACE pg_default;
223
224 ALTER TABLE public.correio_eletronico
225     OWNER to postgres;
226
227
228 -- Table: public.telefone
229
```

```
230 -- DROP TABLE public.telefone;
231
232 CREATE TABLE public.telefone
233 (
234     numero bigint NOT NULL,
235     cod_entidade bigint NOT NULL,
236     ano smallint NOT NULL,
237     ddd smallint,
238     fax boolean,
239     CONSTRAINT telefone_pkey PRIMARY KEY (numero, cod_entidade, ano),
240     CONSTRAINT telefone_cod_entidade_fkey FOREIGN KEY (cod_entidade)
241         REFERENCES public.escola (cod_entidade) MATCH SIMPLE
242         ON UPDATE NO ACTION
243         ON DELETE NO ACTION
244         NOT VALID
245 )
246 WITH (
247     OIDS = FALSE
248 )
249 TABLESPACE pg_default;
250
251 ALTER TABLE public.telefone
252     OWNER to postgres;
```

APÊNDICE C – CONTEXTO POSTGRES

Abaixo está a classe de contexto do BD SQL.

```
{
        public postgresContext()
        {
        }

        public postgresContext(DbContextOptions<postgresContext> options)
                : base(options)
        {
        }

        public virtual DbSet<CensoEscola> CensoEscola { get; set; }
        public virtual DbSet<CorreioEletronico> CorreioEletronico { get; set; }
        public virtual DbSet<Endereco> Endereco { get; set; }
        public virtual DbSet<Escola> Escola { get; set; }
        public virtual DbSet<Estado> Estado { get; set; }
        public virtual DbSet<MantenedoraDaEscola> MantenedoraDaEscola { get; set; }
        public virtual DbSet<Municipio> Municipio { get; set; }
        public virtual DbSet<Regiao> Regiao { get; set; }
        public virtual DbSet<Telefone> Telefone { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
```

```
28          modelBuilder.HasPostgresExtension("adminpack");
29
30          modelBuilder.Entity<CensoEscola>(entity =>
31          {
32                  entity.HasKey(e => new { e.Ano, e.CodEntidade })
33                          .HasName("censo_escola_pkey");
34
35                  entity.ToTable("censo_escola");
36
37                  entity.Property(e => e.Ano).HasColumnName("ano");
38
39                  entity.Property(e => e.CodEntidade).HasColumnName("cod_entidade");
40
41                  entity.Property(e => e.AbreFinalSemana).HasColumnName("abre_final_semana");
42
43                  entity.Property(e => e.Acessibilidade).HasColumnName("acessibilidade");
44
45                  entity.Property(e => e.Aee).HasColumnName("aee");
46
47                  entity.Property(e =>
                    ↪  e.AtividadeComplementar).HasColumnName("atividade_complementar");
48
49                  entity.Property(e => e.DataFimAnoLetivo)
50                          .HasColumnName("data_fim_ano_letivo")
51                          .HasColumnType("date");
```

```
52
53                  entity.Property(e => e.DataInicioAnoLetivo)
54                          .HasColumnName("data_inicio_ano_letivo")
55                          .HasColumnType("date");
56
57                  entity.Property(e =>
                    ↪  e.DependenciaAdministrativa).HasColumnName("dependencia_administrativa");
58
59                  entity.Property(e => e.DependenciasPne).HasColumnName("dependencias_pne");
60
61                  entity.Property(e =>
                    ↪  e.DocumentoRegulamentacao).HasColumnName("documento_regulamentacao");
62
63                  entity.Property(e => e.EducacaoIndigena).HasColumnName("educacao_indigena");
64
65                  entity.Property(e =>
                    ↪  e.EfOrganizadoEmCiclos).HasColumnName("ef_organizado_em_ciclos");
66
67                  entity.Property(e => e.EspacoTurmaPba).HasColumnName("espaco_turma_pba");
68
69                  entity.Property(e => e.IdDependenciaAdm).HasColumnName("id_dependencia_adm");
70
71                  entity.Property(e => e.LinguaIndigena).HasColumnName("lingua_indigena");
72
73                  entity.Property(e => e.LinguaPortuguesa).HasColumnName("lingua_portuguesa");
```

```
74
75                  entity.Property(e => e.ModEducEspecial).HasColumnName("mod_educ_especial");
76
77                  entity.Property(e => e.ModEja).HasColumnName("mod_eja");
78
79                  entity.Property(e => e.ModEnsRegular).HasColumnName("mod_ens_regular");
80
81                  entity.Property(e => e.NumFuncionarios).HasColumnName("num_funcionarios");
82
83                  entity.Property(e => e.NumSalasExistentes).HasColumnName("num_salas_existentes");
84
85                  entity.Property(e => e.NumSalasLeitura).HasColumnName("num_salas_leitura");
86
87                  entity.Property(e => e.NumSalasUsadas).HasColumnName("num_salas_usadas");
88
89                  entity.Property(e => e.Rede).HasColumnName("rede");
90
91                  entity.Property(e => e.SanitariosPne).HasColumnName("sanitarios_pne");
92
93                  entity.Property(e =>
                    ↪  e.SituacaoFuncionamento).HasColumnName("situacao_funcionamento");
94
95                  entity.Property(e => e.Id).HasColumnName("IdBlockchain");
96
97                  entity.HasOne(d => d.CodEntidadeNavigation)
```

```
98                      .WithMany(p => p.CensoEscola)
99                      .HasForeignKey(d => d.CodEntidade)
100                     .OnDelete(DeleteBehavior.ClientSetNull)
101                     .HasConstraintName("censo_escola_cod_entidade_fkey");
102             });
103
104             modelBuilder.Entity<CorreioEletronico>(entity =>
105             {
106                     entity.HasKey(e => new { e.CodEntidade, e.Ano })
107                             .HasName("correio_eletronico_pkey");
108
109                     entity.ToTable("correio_eletronico");
110
111                     entity.Property(e => e.CodEntidade).HasColumnName("cod_entidade");
112
113                     entity.Property(e => e.Ano).HasColumnName("ano");
114
115                     entity.Property(e => e.Email)
116                             .IsRequired()
117                             .HasColumnName("email");
118
119                     entity.Property(e => e.Id).HasColumnName("IdBlockchain");
120
121                     entity.HasOne(d => d.CodEntidadeNavigation)
122                             .WithMany(p => p.CorreioEletronico)
```

```
123                     .HasForeignKey(d => d.CodEntidade)
124                     .OnDelete(DeleteBehavior.ClientSetNull)
125                     .HasConstraintName("correio_eletronico_cod_entidade_fkey");
126             });
127
128             modelBuilder.Entity<Endereco>(entity =>
129             {
130                     entity.HasKey(e => e.CodEndereco)
131                             .HasName("endereco_pkey");
132
133                     entity.ToTable("endereco");
134
135                     entity.Property(e => e.CodEndereco)
136                             .HasColumnName("cod_endereco")
137                             .ValueGeneratedNever();
138
139                     entity.Property(e => e.Bairro)
140                             .IsRequired()
141                             .HasColumnName("bairro");
142
143                     entity.Property(e => e.Cep)
144                             .IsRequired()
145                             .HasColumnName("cep");
146
147                     entity.Property(e => e.CodMunicipio).HasColumnName("cod_municipio");
```

```
148
149                 entity.Property(e => e.Complemento)
150                         .IsRequired()
151                         .HasColumnName("complemento");
152
153                 entity.Property(e => e.Endereco1)
154                         .IsRequired()
155                         .HasColumnName("endereco");
156
157                 entity.Property(e => e.NomeDestrito)
158                         .IsRequired()
159                         .HasColumnName("nome_destrito");
160
161                 entity.Property(e => e.Numero)
162                         .IsRequired()
163                         .HasColumnName("numero");
164
165                 entity.Property(e => e.Id).HasColumnName("IdBlockchain");
166
167                 entity.HasOne(d => d.CodMunicipioNavigation)
168                         .WithMany(p => p.Endereco)
169                         .HasForeignKey(d => d.CodMunicipio)
170                         .OnDelete(DeleteBehavior.ClientSetNull)
171                         .HasConstraintName("endereco_cod_municipio_fkey");
172             });
```

```
173
174             modelBuilder.Entity<Escola>(entity =>
175             {
176                     entity.HasKey(e => e.CodEntidade)
177                             .HasName("escola_pkey");
178
179                     entity.ToTable("escola");
180
181                     entity.Property(e => e.CodEntidade)
182                             .HasColumnName("cod_entidade")
183                             .ValueGeneratedNever();
184
185                     entity.Property(e => e.Categoria).HasColumnName("categoria");
186
187                     entity.Property(e => e.CodEndereco).HasColumnName("cod_endereco");
188
189                     entity.Property(e => e.IdLatitude).HasColumnName("id_latitude");
190
191                     entity.Property(e => e.IdLongitude).HasColumnName("id_longitude");
192
193                     entity.Property(e =>
                        ↪  e.InstituicaoSemFimLucrativo).HasColumnName("instituicao_sem_fim_lucrativo");
194
195                     entity.Property(e => e.Localizacao).HasColumnName("localizacao");
196
```

```
197                 entity.Property(e => e.Nome)
198                         .IsRequired()
199                         .HasColumnName("nome");
200
201                 entity.Property(e => e.Id).HasColumnName("IdBlockchain");
202
203                 entity.HasOne(d => d.CodEnderecoNavigation)
204                         .WithMany(p => p.Escola)
205                         .HasForeignKey(d => d.CodEndereco)
206                         .OnDelete(DeleteBehavior.ClientSetNull)
207                         .HasConstraintName("escola_cod_endereco_fkey");
208         });
209
210         modelBuilder.Entity<Estado>(entity =>
211         {
212                 entity.HasKey(e => e.CodEstado)
213                         .HasName("estado_pkey");
214
215                 entity.ToTable("estado");
216
217                 entity.Property(e => e.CodEstado)
218                         .HasColumnName("cod_estado")
219                         .ValueGeneratedNever();
220
221                 entity.Property(e => e.CodRegiao).HasColumnName("cod_regiao");
```

```
222
223                 entity.Property(e => e.NomeEstado)
224                         .IsRequired()
225                         .HasColumnName("nome_estado");
226
227                 entity.Property(e => e.Uf)
228                         .IsRequired()
229                         .HasColumnName("UF")
230                         .HasMaxLength(2)
231                         .IsFixedLength();
232
233                 entity.Property(e => e.Id).HasColumnName("IdBlockchain");
234
235                 entity.HasOne(d => d.CodRegiaoNavigation)
236                         .WithMany(p => p.Estado)
237                         .HasForeignKey(d => d.CodRegiao)
238                         .OnDelete(DeleteBehavior.ClientSetNull)
239                         .HasConstraintName("estado_cod_regiao_fkey");
240         });
241
242         modelBuilder.Entity<MantenedoraDaEscola>(entity =>
243         {
244                 entity.HasKey(e => e.CodEntidade)
245                         .HasName("mantenedora_da_escola_pkey");
246
```

```
247                 entity.ToTable("mantenedora_da_escola");
248
249                 entity.Property(e => e.CodEntidade)
250                         .HasColumnName("cod_entidade")
251                         .ValueGeneratedNever();
252
253                 entity.Property(e => e.Empresa).HasColumnName("empresa");
254
255                 entity.Property(e => e.Senai).HasColumnName("senai");
256
257                 entity.Property(e => e.Sesc).HasColumnName("sesc");
258
259                 entity.Property(e => e.Sindicato).HasColumnName("sindicato");
260
261                 entity.Property(e => e.SistemsSSesi).HasColumnName("sistems_s_sesi");
262
263                 entity.Property(e => e.Id).HasColumnName("IdBlockchain");
264
265                 entity.HasOne(d => d.CodEntidadeNavigation)
266                         .WithOne(p => p.MantenedoraDaEscola)
267                         .HasForeignKey<MantenedoraDaEscola>(d => d.CodEntidade)
268                         .OnDelete(DeleteBehavior.ClientSetNull)
269                         .HasConstraintName("mantenedora_da_escola_cod_entidade_fkey");
270         });
271
```

```
272             modelBuilder.Entity<Municipio>(entity =>
273             {
274                     entity.HasKey(e => e.CodMunicipio)
275                             .HasName("municipio_pkey");
276
277                     entity.ToTable("municipio");
278
279                     entity.Property(e => e.CodMunicipio)
280                             .HasColumnName("cod_municipio")
281                             .ValueGeneratedNever();
282
283                     entity.Property(e => e.CodEstado).HasColumnName("cod_estado");
284
285                     entity.Property(e => e.NomeMunicipio)
286                             .IsRequired()
287                             .HasColumnName("nome_municipio");
288
289                     entity.Property(e => e.PkCodMunicipioOld).HasColumnName("pk_cod_municipio_old");
290
291                     entity.Property(e => e.Id).HasColumnName("IdBlockchain");
292
293                     entity.HasOne(d => d.CodEstadoNavigation)
294                             .WithMany(p => p.Municipio)
295                             .HasForeignKey(d => d.CodEstado)
296                             .OnDelete(DeleteBehavior.ClientSetNull)
```

```
297                     .HasConstraintName("municipio_cod_estado_fkey");
298             });
299
300             modelBuilder.Entity<Regiao>(entity =>
301             {
302                     entity.HasKey(e => e.CodRegiao)
303                             .HasName("regiao_pkey");
304
305                     entity.ToTable("regiao");
306
307                     entity.Property(e => e.CodRegiao)
308                             .HasColumnName("cod_regiao")
309                             .ValueGeneratedNever();
310
311                     entity.Property(e => e.NomeRegiao)
312                             .IsRequired()
313                             .HasColumnName("nome_regiao");
314
315                     entity.Property(e => e.Id).HasColumnName("IdBlockchain");
316             });
317
318             modelBuilder.Entity<Telefone>(entity =>
319             {
320                     entity.HasKey(e => new { e.Numero, e.CodEntidade, e.Ano })
321                             .HasName("telefone_pkey");
```

```
322
323                     entity.ToTable("telefone");
324
325                     entity.Property(e => e.Numero).HasColumnName("numero");
326
327                     entity.Property(e => e.CodEntidade).HasColumnName("cod_entidade");
328
329                     entity.Property(e => e.Ano).HasColumnName("ano");
330
331                     entity.Property(e => e.Ddd).HasColumnName("ddd");
332
333                     entity.Property(e => e.Fax).HasColumnName("fax");
334
335                     entity.Property(e => e.Id).HasColumnName("IdBlockchain");
336
337                     entity.HasOne(d => d.CodEntidadeNavigation)
338                             .WithMany(p => p.Telefone)
339                             .HasForeignKey(d => d.CodEntidade)
340                             .OnDelete(DeleteBehavior.ClientSetNull)
341                             .HasConstraintName("telefone_cod_entidade_fkey");
342             });
343
344             OnModelCreatingPartial(modelBuilder);
345     }
346
```

```
347                 partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
348         }
349 }
```

APÊNDICE D – ENTIDADE CENSOESCOLA

Abaixo está a classe da Entidade CensoEscola do BD SQL.

```
6           public partial class CensoEscola
7           {
8                   public short? IdDependenciaAdm { get; set; }
9                   public string DependenciaAdministrativa { get; set; }
10                  public string Rede { get; set; }
11                  public DateTime? DataInicioAnoLetivo { get; set; }
12                  public DateTime? DataFimAnoLetivo { get; set; }
13                  public string SituacaoFuncionamento { get; set; }
14                  public bool? EfOrganizadoEmCiclos { get; set; }
15                  public string AtividadeComplementar { get; set; }
16                  public string DocumentoRegulamentacao { get; set; }
17                  public bool? Acessibilidade { get; set; }
18                  public string DependenciasPne { get; set; }
19                  public string SanitariosPne { get; set; }
20                  public string Aee { get; set; }
21                  public long? NumSalasExistentes { get; set; }
22                  public long? NumSalasUsadas { get; set; }
23                  public long? NumSalasLeitura { get; set; }
24                  public long? NumFuncionarios { get; set; }
25                  public bool? EducacaoIndigena { get; set; }
26                  public bool? LinguaIndigena { get; set; }
27                  public bool? LinguaPortuguesa { get; set; }
28                  public bool? EspacoTurmaPba { get; set; }
29                  public bool? AbreFinalSemana { get; set; }
30                  public bool? ModEnsRegular { get; set; }
31                  public bool? ModEducEspecial { get; set; }
32                  public bool? ModEja { get; set; }
33                  public short Ano { get; set; }
34                  public long CodEntidade { get; set; }
35                  public BigInteger? Id { get; set; }
36
37                  public virtual Escola CodEntidadeNavigation { get; set; }
38          }
39  }
```

APÊNDICE E – ENTIDADE CORREIOELETRONICO

Abaixo está a classe da Entidade CorreioEletronico do BD SQL.

```
6           {
7                   public long CodEntidade { get; set; }
8                   public short Ano { get; set; }
9                   public string Email { get; set; }
10                  public BigInteger? Id { get; set; }
11
12                  public virtual Escola CodEntidadeNavigation { get; set; }
13          }
14  }
```

APÊNDICE F – ENTIDADE ENDERECO

Abaixo está a classe da Entidade Endereco do BD SQL.

```
6           public partial class Endereco
7           {
8                   public Endereco()
9                   {
10                          Escola = new HashSet<Escola>();
11                  }
12
13                  public long CodEndereco { get; set; }
14                  public long CodMunicipio { get; set; }
15                  public string Cep { get; set; }
16                  public string NomeDestrito { get; set; }
17                  public string Endereco1 { get; set; }
18                  public string Numero { get; set; }
19                  public string Complemento { get; set; }
20                  public string Bairro { get; set; }
21                  public BigInteger? Id { get; set; }
22
23                  public virtual Municipio CodMunicipioNavigation { get;
                    ↪  set; }
24                  public virtual ICollection<Escola> Escola { get; set; }
25          }
26  }
```

APÊNDICE G – ENTIDADE ESCOLA

Abaixo está a classe da Entidade Escola do BD SQL.

```
6           public partial class Escola
7           {
8                   public Escola()
9                   {
10                          CensoEscola = new HashSet<CensoEscola>();
11                          CorreioEletronico = new
                            ↪  HashSet<CorreioEletronico>();
12                          Telefone = new HashSet<Telefone>();
13                  }
14
15                  public long CodEntidade { get; set; }
16                  public long CodEndereco { get; set; }
17                  public bool? Localizacao { get; set; }
18                  public string Nome { get; set; }
19                  public string Categoria { get; set; }
20                  public string IdLongitude { get; set; }
21                  public string IdLatitude { get; set; }
22                  public string InstituicaoSemFimLucrativo { get; set; }
23                  public BigInteger? Id { get; set; }
24
25                  public virtual Endereco CodEnderecoNavigation { get; set;
                    ↪  }
26                  public virtual MantenedoraDaEscola MantenedoraDaEscola {
                    ↪  get; set; }
27                  public virtual ICollection<CensoEscola> CensoEscola { get;
                    ↪  set; }
28                  public virtual ICollection<CorreioEletronico>
                    ↪  CorreioEletronico { get; set; }
29                  public virtual ICollection<Telefone> Telefone { get; set;
                    ↪  }
30          }
31  }
```

APÊNDICE H – ENTIDADE ESTADO

Abaixo está a classe da Entidade Estado do BD SQL.

```
6           public partial class Estado
7           {
8                   public Estado()
9                   {
10                          Municipio = new HashSet<Municipio>();
11                  }
12
13                  public long CodEstado { get; set; }
14                  public string Uf { get; set; }
15                  public string NomeEstado { get; set; }
16                  public long CodRegiao { get; set; }
17                  public BigInteger? Id { get; set; }
18
19                  public virtual Regiao CodRegiaoNavigation { get; set; }
20                  public virtual ICollection<Municipio> Municipio { get;
                    ↪  set; }
21          }
22  }
```

APÊNDICE I – ENTIDADE MANTENEDORADAESCOLA

Abaixo está a classe da Entidade MantenedoraDaEscola do BD SQL.

```
6           {
7                   public long CodEntidade { get; set; }
8                   public bool Empresa { get; set; }
9                   public bool Sindicato { get; set; }
10                  public bool SistemsSSesi { get; set; }
11                  public bool Senai { get; set; }
12                  public bool Sesc { get; set; }
13                  public BigInteger? Id { get; set; }
14
15                  public virtual Escola CodEntidadeNavigation { get; set; }
16          }
17  }
```

APÊNDICE J – ENTIDADE MUNICIPIO

Abaixo está a classe da Entidade Municipio do BD SQL.

```
6           public partial class Municipio
7           {
8                   public Municipio()
9                   {
10                          Endereco = new HashSet<Endereco>();
11                  }
12
13                  public long CodMunicipio { get; set; }
14                  public long CodEstado { get; set; }
15                  public long PkCodMunicipioOld { get; set; }
16                  public string NomeMunicipio { get; set; }
17                  public BigInteger? Id { get; set; }
18
19                  public virtual Estado CodEstadoNavigation { get; set; }
20                  public virtual ICollection<Endereco> Endereco { get; set;
                    ↪ }
21          }
22  }
```

APÊNDICE K – ENTIDADE REGIAO

Abaixo está a classe da Entidade Regiao do BD SQL.

```
6           public partial class Regiao
7           {
8                   public Regiao()
9                   {
10                          Estado = new HashSet<Estado>();
11                  }
12
13                  public long CodRegiao { get; set; }
14                  public string NomeRegiao { get; set; }
15                  public BigInteger? Id { get; set; }
16
17                  public virtual ICollection<Estado> Estado { get; set; }
18          }
19  }
```

APÊNDICE L – ENTIDADE TELEFONE

Abaixo está a classe da Entidade Telefone do BD SQL.

```
6           {
7                   public long Numero { get; set; }
8                   public long CodEntidade { get; set; }
9                   public short Ano { get; set; }
10                  public short? Ddd { get; set; }
11                  public bool? Fax { get; set; }
12                  public BigInteger? Id { get; set; }
13
14                  public virtual Escola CodEntidadeNavigation { get; set; }
15          }
16  }
```

APÊNDICE M – CONTRATO INTELIGENTE

```
1  contract BlockchainToBD {
2      struct Attributes {
3          uint256 attsId;
4          uint64 numTables;
5          mapping(uint64 => Table) tables;
6      }
7
8      struct Data {
9          bytes32 hash;
10         bool deleted;
11
12         uint64 numTables;
13         mapping(uint64 => Table) tables;
14     }
15
16     struct Table {
17         uint128 numData;
18         mapping(uint128 => Data) dataInTable;
19
20         uint256 numAtt;
21         mapping(uint256 => Attributes) atts;
22
23         uint32 numDB;
24         mapping(uint32 => DB) dbs;
25     }
26
27
28     struct DB {
29         uint64 numTables;
30         mapping(uint64 => Table) tablesInDB;
31
32         bytes32 ipAdd;
33         bytes32 typeOfDB;
34     }
35
36     uint256 numAtt;
37     mapping(uint256 => Attributes) atts;
38     uint32 numDB;
39     mapping(uint32 => DB) dbs;
40     mapping(address => uint32) dbsAddr;
41
42     function verify(uint32 dbId, uint64 tId, uint128 dataId, bytes32
           hash) public view returns (bool check) {
43         if (dbs[dbId].tablesInDB[tId].dataInTable[dataId].hash == hash
               && !dbs[dbId].tablesInDB[tId].dataInTable[dataId].deleted)
```

```
44              return true;
45          return false;
46      }
47
48      function addDB(bytes32 IPAddress) public returns (uint32 dbId) {
49          dbsAddr[msg.sender] = numDB;
50          dbs[numDB].ipAdd = IPAddress;
51      emit DbCreated(numDB, msg.sender);
52          return numDB++;
53      }
54
55      function getIPAddr(address locate) public view returns (bytes32 ip)
            {
56          return dbs[dbsAddr[locate]].ipAdd;
57      }
58
59      function addTable(uint32 dbId) public returns (uint64 tId) {
60          dbs[dbId].tablesInDB[dbs[dbId].numTables].dbs[dbs[dbId].
                tablesInDB[dbs[dbId].numTables].numDB++] = dbs[dbId];
61      emit TableCreated(numDB, dbs[dbId].numTables, msg.sender);
62          return dbs[dbId].numTables++;
63      }
64
65      function addAttribute() public returns (uint256 AttCode) {
66          atts[numAtt].attsId = numAtt;
67      emit AttAdded(numAtt, msg.sender);
68          return numAtt++;
69      }
70
71      function addData(uint32 dbId, uint64 tId, bytes32 dataHash) public
            returns (uint128 Id) {
72          uint128 dataId = dbs[dbId].tablesInDB[tId].numData++;
73          bytes32 prev = dbs[dbId].tablesInDB[tId].dataInTable[dataId].
                hash;
74          dbs[dbId].tablesInDB[tId].dataInTable[dataId].hash = dataHash;
75          dbs[dbId].tablesInDB[tId].dataInTable[dataId].deleted = false;
76          emit DataModified(dbId, tId, dataId, msg.sender, prev, dataHash)
                ;
77          return dataId;
78      }
79
80      function linkDataToAtt(uint32 dbId, uint64 tId, uint256 attId)
            public {
81          atts[attId].tables[atts[attId].numTables++] = dbs[dbId].
                tablesInDB[tId];
82          dbs[dbId].tablesInDB[tId].atts[dbs[dbId].tablesInDB[tId].numAtt
                ++] = atts[attId];
```

```
83      }
84
85      function updateData(uint32 dbId, uint64 tId, uint128 dId, bytes32
            dataHash, bytes32 prevHash) public returns (bool updated) {
86          if (prevHash == dbs[dbId].tablesInDB[tId].dataInTable[dId].hash
                && !dbs[dbId].tablesInDB[tId].dataInTable[dId].deleted) {
87              dbs[dbId].tablesInDB[tId].dataInTable[dId].hash = dataHash;
88              for(uint256 i = 0; i < dbs[dbId].tablesInDB[tId].numAtt; i
                    ++) {
89                  uint256 attId = dbs[dbId].tablesInDB[tId].atts[i].attsId
                        ;
90
91                  for (uint64 k = 0; k < atts[attId].numTables; k++)
92            for (uint32 l = 0; l < atts[attId].tables[k].numDB; l++)
93                          emit DataModified(l, k, dId, msg.sender,
                                prevHash, dataHash);
94              }
95
96              return true;
97          }
98          return false;
99      }
100
101     function deleteData(uint32 dbId, uint64 tId, uint128 dId, bytes32
            dataHash) public returns (bool deleted) {
102         if (dataHash == dbs[dbId].tablesInDB[tId].dataInTable[dId].hash
                && !dbs[dbId].tablesInDB[tId].dataInTable[dId].deleted) {
103             dbs[dbId].tablesInDB[tId].dataInTable[dId].deleted = true;
104
105             emit DataDeleted(dbId, tId, dId, msg.sender);
106
107             return true;
108         }
109         return false;
110     }
111
112     function deleteDataFromAll(uint32 dbId, uint64 tId, uint128 dId,
            bytes32 dataHash) public returns (bool deleted) {
113         if (dataHash == dbs[dbId].tablesInDB[tId].dataInTable[dId].hash
                && !dbs[dbId].tablesInDB[tId].dataInTable[dId].deleted) {
114             dbs[dbId].tablesInDB[tId].dataInTable[dId].deleted = true;
115
116             for(uint256 i = 0; i < dbs[dbId].tablesInDB[tId].numAtt; i
                    ++) {
117                 uint256 attId = dbs[dbId].tablesInDB[tId].atts[i].attsId
                        ;
118
```

```
119                 for (uint64 k = 0; k < atts[attId].numTables; k++)
120           for (uint32 l = 0; l < atts[attId].tables[k].numDB; l++)
121                       emit DataDeleted(l, k, dId, msg.sender);
122             }
123             return true;
124         }
125         return false;
126     }
127
128     event DataModified (
129         uint32 indexed dbId,
130         uint64 tId,
131         uint128 dId,
132         address sender,
133         bytes32 prevHash,
134         bytes32 nextHash
135     );
136     event DataDeleted (
137         uint32 indexed dbId,
138         uint64 tId,
139         uint128 dId,
140         address sender
141     );
142     event TableCreated (
143         uint32 indexed dbId,
144         uint64 tId,
145         address sender
146     );
147     event DbCreated (
148         uint32 indexed dbId,
149         address sender
150     );
151     event AttAdded (
152         uint256 indexed attId,
153         address sender
154     );
155
156 }
```

APÊNDICE N – CONTROLADOR CENSOESCOLA

```
8  namespace API.SQL.Controllers
9  {
10         [Route("api/[controller]")]
11         [ApiController]
12         public class CensoEscolasController : ControllerBase
13         {
14                 private readonly postgresContext _context;
15
16                 public CensoEscolasController(postgresContext context)
17                 {
18                         _context = context;
19                 }
20
21                 // GET: api/CensoEscolas
22                 [HttpGet]
23                 public async Task<ActionResult<IEnumerable<CensoEscola>>> GetCensoEscola()
24                 {
25                         return await _context.CensoEscola.ToListAsync();
26                 }
27
28                 // GET: api/CensoEscolas/2018/101010
29                 [HttpGet("{ano}/{id}")]
30                 public async Task<ActionResult<CensoEscola>> GetCensoEscola(short ano, long id)
```

```
31          {
32                  var censoEscola = await _context.CensoEscola.Where(ce => ce.Ano == ano && ce.CodEntidade
                    ↪  == id).FirstAsync();
33
34                  if (censoEscola == null)
35                  {
36                          return NotFound();
37                  }
38
39                  return censoEscola;
40          }
41
42          // PUT: api/CensoEscolas/2018/101010
43          // To protect from overposting attacks, please enable the specific properties you want to bind to,
                ↪  for
44          // more details see https://aka.ms/RazorPagesCRUD.
45          [HttpPut("{ano}/{id}")]
46          public async Task<IActionResult> PutCensoEscola(short ano, long id, CensoEscola censoEscola)
47          {
48                  if (id != censoEscola.CodEntidade || ano != censoEscola.Ano)
49                  {
50                          return BadRequest();
51                  }
52
53                  _context.Entry(censoEscola).State = EntityState.Modified;
```

```
54
55              try
56              {
57                      await _context.SaveChangesAsync();
58              }
59              catch (DbUpdateConcurrencyException)
60              {
61                      if (!CensoEscolaExists(ano, id))
62                      {
63                              return NotFound();
64                      }
65                      else
66                      {
67                              throw;
68                      }
69              }
70
71              return NoContent();
72          }
73
74          // POST: api/CensoEscolas
75          // To protect from overposting attacks, please enable the specific properties you want to bind to,
            ↪ for
76          // more details see https://aka.ms/RazorPagesCRUD.
77          [HttpPost]
```

```
78          public async Task<ActionResult<CensoEscola>> PostCensoEscola(CensoEscola censoEscola)
79          {
80                  _context.CensoEscola.Add(censoEscola);
81                  try
82                  {
83                          await _context.SaveChangesAsync();
84                  }
85                  catch (DbUpdateException)
86                  {
87                          if (CensoEscolaExists(censoEscola.Ano, censoEscola.CodEntidade))
88                          {
89                                  return Conflict();
90                          }
91                          else
92                          {
93                                  throw;
94                          }
95                  }
96
97                  return CreatedAtAction("GetCensoEscola", new { ano = censoEscola.Ano, id = censoEscola.Ano
                    ↪ }, censoEscola);
98          }
99
100         // DELETE: api/CensoEscolas/2019/101010
101         [HttpDelete("{ano}/{id}")]
```

```
102             public async Task<ActionResult<CensoEscola>> DeleteCensoEscola(short ano, long id)
103             {
104                     var censoEscola = await _context.CensoEscola.Where(ce => ce.Ano == ano && ce.CodEntidade
                        ↪  == id).FirstAsync();
105                     if (censoEscola == null)
106                     {
107                             return NotFound();
108                     }
109
110                     _context.CensoEscola.Remove(censoEscola);
111                     await _context.SaveChangesAsync();
112
113                     return censoEscola;
114             }
115
116             private bool CensoEscolaExists(short ano, long id)
117             {
118                     return _context.CensoEscola.Any(e => e.Ano == ano && e.CodEntidade == id);
119             }
120     }
121 }
```

APÊNDICE O – CONTROLADOR TELEFONE

```
8  namespace API.SQL.Controllers
9  {
10         [Route("api/[controller]")]
11         [ApiController]
12         public class TelefonesController : ControllerBase
13         {
14                 private readonly postgresContext _context;
15
16                 public TelefonesController(postgresContext context)
17                 {
18                         _context = context;
19                 }
20
21                 // GET: api/Telefones
22                 [HttpGet]
23                 public async Task<ActionResult<IEnumerable<Telefone>>> GetTelefone()
24                 {
25                         return await _context.Telefone.ToListAsync();
26                 }
27
28                 // GET: api/Telefones/2018/101010
29                 [HttpGet("{ano}/{id}")]
30                 public async Task<ActionResult<IEnumerable<Telefone>>> GetTelefone(short ano, long id)
```

```
31          {
32                  return await _context.Telefone.Where(ce => ce.Ano == ano && ce.CodEntidade ==
                    ↪  id).ToListAsync();
33          }
34
35          // GET: api/Telefones/33333333/2018/101010
36          [HttpGet("{num}/{ano}/{id}")]
37          public async Task<ActionResult<Telefone>> GetTelefone(long num, short ano, long id)
38          {
39                  var telefone = await _context.Telefone.Where(ce => ce.Numero == num && ce.Ano == ano &&
                    ↪  ce.CodEntidade == id).FirstAsync();
40
41                  if (telefone == null)
42                  {
43                          return NotFound();
44                  }
45
46                  return telefone;
47          }
48
49          // PUT: api/Telefones/33333333/2018/101010
50          // To protect from overposting attacks, please enable the specific properties you want to bind to,
                ↪  for
51          // more details see https://aka.ms/RazorPagesCRUD.
52          [HttpPut("{num}/{ano}/{id}")]
```

```
53          public async Task<IActionResult> PutTelefone(long num, short ano, long id, Telefone telefone)
54          {
55                  if (id != telefone.CodEntidade || ano != telefone.Ano || num != telefone.Numero)
56                  {
57                          return BadRequest();
58                  }
59
60                  _context.Entry(telefone).State = EntityState.Modified;
61
62                  try
63                  {
64                          await _context.SaveChangesAsync();
65                  }
66                  catch (DbUpdateConcurrencyException)
67                  {
68                          if (!TelefoneExists(num, ano, id))
69                          {
70                                  return NotFound();
71                          }
72                          else
73                          {
74                                  throw;
75                          }
76                  }
77
```

```
78              return NoContent();
79          }
80
81          // POST: api/Telefones
82          // To protect from overposting attacks, please enable the specific properties you want to bind to,
            ↪  for
83          // more details see https://aka.ms/RazorPagesCRUD.
84          [HttpPost]
85          public async Task<ActionResult<Telefone>> PostTelefone(Telefone telefone)
86          {
87              _context.Telefone.Add(telefone);
88              try
89              {
90                  await _context.SaveChangesAsync();
91              }
92              catch (DbUpdateException)
93              {
94                  if (TelefoneExists(telefone.Numero, telefone.Ano, telefone.CodEntidade))
95                  {
96                      return Conflict();
97                  }
98                  else
99                  {
100                     throw;
101                 }
```

```
102             }
103
104             return CreatedAtAction("GetTelefone", new { num = telefone.Numero, ano = telefone.Ano, id
                ↪  = telefone.CodEntidade }, telefone);
105         }
106
107         // DELETE: api/Telefones/33333333/2018/101010
108         [HttpDelete("{num}/{ano}/{id}")]
109         public async Task<ActionResult<Telefone>> DeleteTelefone(long num, short ano, long id)
110         {
111             var telefone = await _context.Telefone.Where(ce => ce.Numero == num && ce.Ano == ano &&
                ↪  ce.CodEntidade == id).FirstAsync();
112             if (telefone == null)
113             {
114                 return NotFound();
115             }
116
117             _context.Telefone.Remove(telefone);
118             await _context.SaveChangesAsync();
119
120             return telefone;
121         }
122
123         private bool TelefoneExists(long num, short ano, long id)
124         {
```

```
125                     return _context.Telefone.Any(e => e.Numero == num && e.Ano == ano && e.CodEntidade == id);
126                 }
127         }
128 }
```

APÊNDICE P – CONTROLADOR CORREIOELETRONICO

```
8  namespace API.SQL.Controllers
9  {
10         [Route("api/[controller]")]
11         [ApiController]
12         public class CorreioEletronicoController : ControllerBase
13         {
14                 private readonly postgresContext _context;
15
16                 public CorreioEletronicoController(postgresContext context)
17                 {
18                         _context = context;
19                 }
20
21                 // GET: api/CorreioEletronicoes
22                 [HttpGet]
23                 public async Task<ActionResult<IEnumerable<CorreioEletronico>>> GetCorreioEletronico()
24                 {
25                         return await _context.CorreioEletronico.ToListAsync();
26                 }
27
28                 // GET: api/CorreioEletronicoes/5
29                 [HttpGet("{ano}/{id}")]
30                 public async Task<ActionResult<CorreioEletronico>> GetCorreioEletronico(short ano, long id)
```

```
31          {
32                  var correioEletronico = await _context.CorreioEletronico.Where(ce => ce.Ano == ano &&
                    ↪  ce.CodEntidade == id).FirstAsync();
33
34                  if (correioEletronico == null)
35                  {
36                          return NotFound();
37                  }
38
39                  return correioEletronico;
40          }
41
42          // PUT: api/CorreioEletronicoes/5
43          // To protect from overposting attacks, please enable the specific properties you want to bind to,
                ↪  for
44          // more details see https://aka.ms/RazorPagesCRUD.
45          [HttpPut("{ano}/{id}")]
46          public async Task<IActionResult> PutCorreioEletronico(short ano, long id, CorreioEletronico
                ↪  correioEletronico)
47          {
48                  if (id != correioEletronico.CodEntidade || ano != correioEletronico.Ano)
49                  {
50                          return BadRequest();
51                  }
52
```

```
53              _context.Entry(correioEletronico).State = EntityState.Modified;
54
55              try
56              {
57                      await _context.SaveChangesAsync();
58              }
59              catch (DbUpdateConcurrencyException)
60              {
61                      if (!CorreioEletronicoExists(ano, id))
62                      {
63                              return NotFound();
64                      }
65                      else
66                      {
67                              throw;
68                      }
69              }
70
71              return NoContent();
72      }
73
74      // POST: api/CorreioEletronicoes
75      // To protect from overposting attacks, please enable the specific properties you want to bind to,
        ↪  for
76      // more details see https://aka.ms/RazorPagesCRUD.
```

```
77          [HttpPost]
78          public async Task<ActionResult<CorreioEletronico>> PostCorreioEletronico(CorreioEletronico
            ↪ correioEletronico)
79          {
80                  _context.CorreioEletronico.Add(correioEletronico);
81                  try
82                  {
83                          await _context.SaveChangesAsync();
84                  }
85                  catch (DbUpdateException)
86                  {
87                          if (CorreioEletronicoExists(correioEletronico.Ano, correioEletronico.CodEntidade))
88                          {
89                                  return Conflict();
90                          }
91                          else
92                          {
93                                  throw;
94                          }
95                  }
96
97                  return CreatedAtAction("GetCorreioEletronico", new { ano = correioEletronico.Ano, id =
                    ↪ correioEletronico.CodEntidade }, correioEletronico);
98          }
99
```

```
100             // DELETE: api/CorreioEletronicoes/5
101             [HttpDelete("{ano}/{id}")]
102             public async Task<ActionResult<CorreioEletronico>> DeleteCorreioEletronico(short ano, long id)
103             {
104                     var correioEletronico = await _context.CorreioEletronico.Where(ce => ce.Ano == ano &&
                        ↪  ce.CodEntidade == id).FirstAsync();
105                     if (correioEletronico == null)
106                     {
107                             return NotFound();
108                     }
109
110                     _context.CorreioEletronico.Remove(correioEletronico);
111                     await _context.SaveChangesAsync();
112
113                     return correioEletronico;
114             }
115
116             private bool CorreioEletronicoExists(short ano, long id)
117             {
118                     return _context.CorreioEletronico.Any(e => e.CodEntidade == id && e.Ano == ano);
119             }
120     }
121 }
```

APÊNDICE Q – EVENT POOLER

```
10 namespace API.Blockchain
11 {
12         public class EventPool
13         {
14                 public Event<DataModifiedEventDTO> eventDataModifiedLog;
15                 public List<EventLog<DataModifiedEventDTO>> ListEventDataModifiedLog { get; set; }
16                 public Event<DataDeletedEventDTO> eventDataDeletedLog;
17                 public List<EventLog<DataDeletedEventDTO>> ListEventDataDeletedLog { get; set; }
18
19                 public EventHandlerBlockchain Handler { get; set; }
20
21
22                 public EventPool(Web3 web3)
23                 {
24                         Handler = new EventHandlerBlockchain();
25                         eventDataModifiedLog = web3.Eth.GetEvent<DataModifiedEventDTO>("DataModified");
26                         eventDataDeletedLog = web3.Eth.GetEvent<DataDeletedEventDTO>("DataDeleted");
27
28                         _ = EventPooling();
29                 }
30
31                 public async Task EventPooling()
32                 {
```

```
33  var filter = eventDataModifiedLog.CreateFilterInput();
34  var filter2 = eventDataDeletedLog.CreateFilterInput();
35
36  Dictionary<uint, HashSet<ulong>> dbAndTables = new Dictionary<uint, HashSet<ulong>>();
37  Dictionary<ulong, HashSet<BigInteger>> tableAndData = new Dictionary<ulong,
    ↪  HashSet<BigInteger>>();
38  Dictionary<BigInteger, (byte[], byte[])> dataAndHashs = new Dictionary<BigInteger, (byte[],
    ↪  byte[])>();
39
40  while (true)
41  {
42          System.Threading.Thread.Sleep(5000);
43          var log = await eventDataModifiedLog.GetAllChanges(filter);
44
45          if (ListEventDataModifiedLog.Count != log.Count)
46          {
47                  var dif = ListEventDataModifiedLog.Intersect(log);
48
49                  var loop = dif.Except(log);
50
51                  string sender = null;
52                  foreach (var item in loop)
53                  {
54                          dbAndTables.TryAdd(item.Event.DbId, new HashSet<ulong>());
55                          dbAndTables[item.Event.DbId].Add(item.Event.TId);
```

```
56                          tableAndData.TryAdd(item.Event.TId, new HashSet<BigInteger>());
57                          tableAndData[item.Event.TId].Add(item.Event.DId);
58                          dataAndHashs.TryAdd(item.Event.DId, (item.Event.PrevHash,
                            ↪  item.Event.NextHash));
59                          sender = item.Event.Sender;
60                  }
61                  await Handler.DataModifiedEventHandler(dbAndTables, tableAndData,
                    ↪  dataAndHashs);
62
63                  dbAndTables.Clear();
64                  tableAndData.Clear();
65                  dataAndHashs.Clear();
66          }
67
68
69          var log2 = await eventDataDeletedLog.GetAllChanges(filter2);
70
71          if (ListEventDataDeletedLog.Count != log2.Count)
72          {
73                  var dif = ListEventDataDeletedLog.Intersect(log2);
74
75                  var loop = dif.Except(log2);
76
77                  foreach (var item in loop)
78                  {
```

```
79                                          dbAndTables.TryAdd(item.Event.DbId, new HashSet<ulong>());
80                                          dbAndTables[item.Event.DbId].Add(item.Event.TId);
81                                          tableAndData.TryAdd(item.Event.TId, new HashSet<BigInteger>());
82                                          tableAndData[item.Event.TId].Add(item.Event.DId);
83                                      }
84                                      await Handler.DataDeletedEventHandler(dbAndTables, tableAndData);
85
86                                      dbAndTables.Clear();
87                                      tableAndData.Clear();
88                                  }
89                              }
90                          }
91
92                  }
93  }
```

APÊNDICE R – STARTUP API BLOCKCHAIN

```
14  public class Startup
15  {
16          public Startup(IConfiguration configuration)
17          {
18                  Configuration = configuration;
19          }
20
21          public IConfiguration Configuration { get; }
22          public static BlockchainToBDService blockchainService { get; set; }
23
24          // This method gets called by the runtime. Use this method to add services to the container.
25          public async void ConfigureServices(IServiceCollection services)
26          {
27                  var account = new Account(Configuration.GetValue<string>("Ethereum:private"));
28                  var web3 = new Web3(account, Configuration.GetValue<string>("Ethereum:url"));
29                  var deploymentMessage = new BlockchainToBDDeployment();
30
31                  var deploymentHandler = web3.Eth.GetContractDeploymentHandler<BlockchainToBDDeployment>();
32                  var transactionReceipt = await
                    ↪ deploymentHandler.SendRequestAndWaitForReceiptAsync(deploymentMessage);
33                  var contractAddress = transactionReceipt.ContractAddress;
34
35                  blockchainService = new BlockchainToBDService(web3, contractAddress);
```

```
36
37              _ = new EventPool(web3);
38
39              services.AddControllers();
40          }
41
42          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
43          public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
44          {
45              if (env.IsDevelopment())
46              {
47                  app.UseDeveloperExceptionPage();
48              }
49
50              app.UseHttpsRedirection();
51
52              app.UseRouting();
53
54              app.UseAuthorization();
55
56              app.UseEndpoints(endpoints =>
57              {
58                  endpoints.MapControllers();
59              });
60          }
```

```
}

    }
}
```

APÊNDICE S – CONTROLADOR DE ATRIBUTOS

```
9  namespace API.Blockchain.Controllers
10 {
11         [Route("api/[controller]")]
12         [ApiController]
13         public class AttributesController : ControllerBase
14         {
15                 public BlockchainToBDService Service { get; set; }
16                 public AttributesController(BlockchainToBDService service)
17                 {
18                         Service = service;
19                 }
20
21                 // POST: api/Attributes
22                 [HttpPost]
23                 public async Task<BigInteger> Post(AddAttributeFunction att)
24                 {
25                         var receipt = await Service.AddAttributeRequestAndWaitForReceiptAsync(att);
26                         var attEvent = receipt.DecodeAllEvents<AttAddedEventDTO>();
27
28                         return attEvent.LastOrDefault().Event.AttId;
29                 }
30
31                 // PUT: api/Attributes/5
```

```
32                  [HttpPut]
33                  public async Task Put(LinkDataToAttFunction linker)
34                  {
35                          var _ = await Service.LinkDataToAttRequestAndWaitForReceiptAsync(linker);
36                  }
37
38          }
39  }
```

APÊNDICE T – CONTROLADOR DE BANCO DE DADOS

```
8  namespace API.Blockchain.Controllers
9  {
10         [Route("api/[controller]")]
11         [ApiController]
12         public class DatabaseController : ControllerBase
13         {
14                 public BlockchainToBDService Service { get; set; }
15                 public DatabaseController(BlockchainToBDService service)
16                 {
17                         Service = service;
18                 }
19
20                 // GET: api/DataBase/5
21                 [HttpGet("{id}")]
22                 public async Task<byte[]> Get(string id)
23                 {
24                         return await Service.GetIPAddrQueryAsync(id);
25                 }
26
27                 // POST: api/DataBase
28                 [HttpPost]
29                 public async Task<uint> PostAsync(AddDBFunction addDB)
30                 {
```

```
31                          var receipt = await Service.AddDBRequestAndWaitForReceiptAsync(addDB);
32                          var AddDbEvent = receipt.DecodeAllEvents<DbCreatedEventDTO>();
33
34                          return AddDbEvent.LastOrDefault().Event.DbId;
35                      }
36              }
37  }
```

APÊNDICE U – CONTROLADOR DE DADOS

```
9  namespace API.Blockchain.Controllers
10 {
11         [Route("api/[controller]")]
12         [ApiController]
13         public class DataController : ControllerBase
14         {
15                 public BlockchainToBDService _service { get; set; }
16                 public DataController(BlockchainToBDService service)
17                 {
18                         _service = service;
19                 }
20
21                 // GET: api/Data/db/table/data/hash
22                 [HttpGet("{dbId}/{tId}/{dId}/{h}")]
23                 public async Task<bool> Get(uint dbId, ulong tId, BigInteger dId, byte[] h)
24                 {
25                         return await _service.VerifyQueryAsync(dbId, tId, dId, h);
26                 }
27
28                 // POST: api/Data
29                 [HttpPost]
30                 public async Task<BigInteger> Post(AddDataFunction Data)
31                 {
```

```
32              var receipt = await _service.AddDataRequestAndWaitForReceiptAsync(Data);
33              var AddDataEvent = receipt.DecodeAllEvents<DataModifiedEventDTO>();
34
35              return AddDataEvent.LastOrDefault().Event.DId;
36          }
37
38          // PUT: api/Data
39          [HttpPut]
40          public async Task<byte[]> Put(UpdateDataFunction update)
41          {
42              var receipt = await _service.UpdateDataRequestAndWaitForReceiptAsync(update);
43              var AddDataEvent = receipt.DecodeAllEvents<DataModifiedEventDTO>();
44
45              return AddDataEvent.LastOrDefault().Event.NextHash;
46          }
47
48          // DELETE: api/Data/del/
49          [HttpDelete("del/")]
50          public async Task<bool> Delete(uint dbId, ulong tId, BigInteger dId, byte[] h)
51          {
52              var receipt = await _service.DeleteDataRequestAndWaitForReceiptAsync(dbId, tId, dId, h);
53              var DelDataEvent = receipt.DecodeAllEvents<DataDeletedEventDTO>();
54
55              return DelDataEvent.LastOrDefault().Event.DId == dId;
56          }
```

```
57
58              // DELETE: api/Data/all/
59              [HttpDelete("all/")]
60              public async Task<bool> DeleteAll(uint dbId, ulong tId, BigInteger dId, byte[] h)
61              {
62                      var receipt = await _service.DeleteDataFromAllRequestAndWaitForReceiptAsync(dbId, tId, dId,
                        ↪ h);
63                      var DelDataEvent = receipt.DecodeAllEvents<DataDeletedEventDTO>();
64
65                      return DelDataEvent.LastOrDefault().Event.DId == dId;
66              }
67      }
68  }
```

APÊNDICE V – CONTROLADOR DE TABELAS

```
8  namespace API.Blockchain.Controllers
9  {
10         [Route("api/[controller]")]
11         [ApiController]
12         public class TablesController : ControllerBase
13         {
14                 public BlockchainToBDService _service { get; set; }
15                 public TablesController(BlockchainToBDService service)
16                 {
17                         _service = service;
18                 }
19
20                 // POST: api/Tables
21                 [HttpPost]
22                 public async Task<ulong> Post(AddTableFunction addTable)
23                 {
24                         var receipt = await _service.AddTableRequestAndWaitForReceiptAsync(addTable);
25                         var addTableEvent = receipt.DecodeAllEvents<TableCreatedEventDTO>();
26
27                         return addTableEvent.LastOrDefault().Event.TId;
28                 }
29         }
30 }
```

APÊNDICE W – SETTINGS MONGODB

```
1  namespace SharedLibrary.Context.Custom
2  {
3          public class MongoDBSettings : IMongoDBSettings
4          {
5                  public string ExtrasCollectionName { get; set; }
6                  public string CollectionString { get; set; }
7                  public string DatabaseName { get; set; }
8          }
9          public interface IMongoDBSettings
10         {
11                 public string ExtrasCollectionName { get; set; }
12                 public string CollectionString { get; set; }
13                 public string DatabaseName { get; set; }
14         }
15 }
```

APÊNDICE X – CONTEXTO MONGO

```
6  namespace SharedLibrary.Context.Custom
7  {
8          public class ExtrasDaEscolaContext
9          {
10                 private readonly IMongoCollection<ExtrasDaEscola> _extras;
11
12                 public ExtrasDaEscolaContext(IMongoDBSettings settings)
13                 {
14                         var client = new MongoClient(settings.CollectionString);
15                         var database = client.GetDatabase(settings.DatabaseName);
16
17                         _extras = database.GetCollection<ExtrasDaEscola>(settings.ExtrasCollectionName);
18                 }
19
20                 public List<ExtrasDaEscola> Get() =>
21                         _extras.Find(extra => true).ToList();
22                 public ExtrasDaEscola Get(Indexer id) =>
23                         _extras.Find<ExtrasDaEscola>(extra => extra.ID == id).FirstOrDefault();
24                 public void Upsert(Indexer iD, ExtrasDaEscola extraNovo) =>
25                         _extras.ReplaceOne(extra => extra.ID == extraNovo.ID, extraNovo, new UpdateOptions {
                           ↪  IsUpsert = true });
26                 public void Remove(ExtrasDaEscola extraIn) =>
27                         _extras.DeleteOne(extra => extra.ID == extraIn.ID);
```

```
28                  public void Remove(Indexer extraId) =>
29                          _extras.DeleteOne(extra => extra.ID == extraId);
30          }
31  }
```

APÊNDICE Y – ENTIDADES DO MONGODB

```
1  using MongoDB.Bson.Serialization.Attributes;
2  using System.Collections.Generic;
3  using System.Numerics;
4
5  namespace SharedLibrary.Entities.Custom
6  {
7          public class ExtrasDaEscola
8          {
9                  public class Tipo_Educacional
10                 {
11                         public int Cod_TE { get; set; }
12                         public string TipoEducacional { get; set; }
13                 }
14                 public class Numero_Matriculas
15                 {
16                         public Tipo_Educacional TE { get; set; }
17                         public string Especificacao { get; set; }
18                         public int Numero_De_Matriculas { get; set; }
19                 }
20                 public class MediaPorAlunoPorEscola
21                 {
22                         public Tipo_Educacional Cod_TE { get; set; }
23                         public int Serie { get; set; }
24                         public double Media { get; set; }
25                 }
26                 public class EquipamentosDaEscola
27                 {
28                         public string Nome_Equip { get; set; }
29                         public short Numero_De_Equip { get; set; }
30                 }
31                 public class MateriaisDidaticosEspecificos
32                 {
33                         public bool MaterialEspecificoNaoUtiliza { get;
                           ↪ set; }
34                         public bool MaterialEspecificoQuilombola { get;
                           ↪ set; }
```

```
35                          public bool MaterialEspecificoIndigena { get; set;
                            ↪ }
36                  }
37                  public class EspecificacaoEscolaPrivada
38                  {
39                          public bool EscolaEFilantropica { get; set; }
40                          public int NumCNPJEscolaPrivada { get; set; }
41                  }
42                  public class Indexer
43                  {
44                          public long Cod_Entidade { get; set; }
45                          public short Ano { get; set; }
46                  }
47
48                  [BsonId]
49                  public Indexer ID { get; set; }
50                  public List<Numero_Matriculas> Matriculas { get; set; }
51                  public List<MediaPorAlunoPorEscola> Medias { get; set; }
52                  public List<EquipamentosDaEscola> Equipamentos { get; set;
                    ↪ }
53                  public MateriaisDidaticosEspecificos MateriaisEspecificos
                    ↪ { get; set; }
54                  public List<string> DependenciasDaEscola { get; set; }
55                  public int NumCNPJUnidadeExecutora { get; set; }
56                  public string ServicosDaEscola { get; set; }
57                  public EspecificacaoEscolaPrivada EscolaPrivada { get;
                    ↪ set; }
58                  public BigInteger Id { get; set; }
59          }
60  }
```

APÊNDICE Z – CONTROLADOR MONGODB

```
7  namespace API.Mongo.Controllers
8  {
9          [Route("api/[controller]")]
10         [ApiController]
11         public class MongoController : ControllerBase
12         {
13                 public readonly ExtrasDaEscolaContext _extrasContext;
14
15                 public MongoController(ExtrasDaEscolaContext context)
16                 {
17                         _extrasContext = context;
18                 }
19
20                 [HttpGet]
21                 public ActionResult<List<ExtrasDaEscola>> Get() =>
22                         _extrasContext.Get();
23
24                 [HttpGet("{ano}/{cod}")]
25                 public ActionResult<ExtrasDaEscola> GetExtra(short ano, long id)
26                 {
27                         Indexer ind = new Indexer { Ano = ano, Cod_Entidade = id };
28                         var extra = _extrasContext.Get(ind);
29
```

```
30              if (extra == null)
31                      return NotFound();
32
33              return extra;
34      }
35
36      [HttpPost]
37      public ActionResult<ExtrasDaEscola> PostExtra(ExtrasDaEscola extra)
38      {
39              _extrasContext.Upsert(extra.ID, extra);
40
41              return CreatedAtRoute("GetExtra", new { id = extra.ID, extra });
42      }
43
44      [HttpPut]
45      public IActionResult PutExtra(ExtrasDaEscola extra)
46      {
47              var ex = _extrasContext.Get(extra.ID);
48
49              if (ex == null)
50              {
51                      return NotFound();
52              }
53
54              _extrasContext.Upsert(extra.ID, extra);
```

```
55
56                      return NoContent();
57              }
58
59              [HttpDelete("{ano}/{cod}")]
60              public IActionResult DeleteExtra(short ano, long id)
61              {
62                      Indexer ind = new Indexer { Ano = ano, Cod_Entidade = id };
63                      var extra = _extrasContext.Get(ind);
64
65                      if (extra == null)
66                              return NotFound();
67
68                      _extrasContext.Remove(extra.ID);
69
70                      return NoContent();
71              }
72      }
73  }
```

```
11  public class EventHandlerBlockchain
12  {
13          public async Task DataModifiedEventHandler(Dictionary<uint, HashSet<ulong>> dbAndTables,
14                  Dictionary<ulong, HashSet<BigInteger>> tableAndData,
15                  Dictionary<BigInteger, (byte[], byte[])> dataAndHashs)
16          {
17                  foreach (var db in dbAndTables)
18                  {
19                          foreach (var table in db.Value)
20                          {
21                                  var t = Globals.TablesTypes[table].Name;
22                                  foreach (var data in tableAndData.GetValueOrDefault(table))
23                                  {
24                                          switch (t)
25                                          {
26                                                  case nameof(CensoEscola):
27                                                          CensoEscolasController cont = new
                                                            ↪  CensoEscolasController();
28                                                          var gett = await cont.GetCensoEscola();
29                                                          var listCE = gett.Value;
30                                                          CensoEscola idd = null;
31                                                          foreach (var item in listCE)
32                                                          {
```

```
33                      idd = item;
34                      if (item.Id == data)
35                              break;
36              }
37              var putt = await cont.PutCensoEscola(idd.Ano,
                ↪  idd.CodEntidade, idd);
38              break;
39      case nameof(CorreioEletronico):
40              CorreioEletronicoController cont2 = new
                ↪  CorreioEletronicoController();
41              var gett2 = await cont2.GetCorreioEletronico();
42              var listCE2 = gett2.Value;
43              CorreioEletronico idd2 = null;
44              foreach (var item in listCE2)
45              {
46                      idd2 = item;
47                      if (item.Id == data)
48                              break;
49              }
50              var putt2 = await
                ↪  cont2.PutCorreioEletronico(idd2.Ano,
                ↪  idd2.CodEntidade, idd2);
51              break;
52      case nameof(Endereco):
```

```
53                  EnderecosController cont3 = new
                    ↪  EnderecosController();
54                  var gett3 = await cont3.GetEndereco();
55                  var listCE3 = gett3.Value;
56                  Endereco idd3 = null;
57                  foreach (var item in listCE3)
58                  {
59                          idd3 = item;
60                          if (item.Id == data)
61                                  break;
62                  }
63                  var putt3 = await
                    ↪  cont3.PutEndereco(idd3.CodEndereco, idd3);
64                  break;
65          case nameof(Escola):
66                  EscolasController cont4 = new EscolasController();
67                  var gett4 = await cont4.GetEscola();
68                  var listCE4 = gett4.Value;
69                  Escola idd4 = null;
70                  foreach (var item in listCE4)
71                  {
72                          idd4 = item;
73                          if (item.Id == data)
74                                  break;
75                  }
```

```
76                  var putt4 = await cont4.PutEscola(idd4.CodEntidade,
                    ↪  idd4);
77                  break;
78          case nameof(Estado):
79                  EstadosController cont5 = new EstadosController();
80                  var gett5 = await cont5.GetEstado();
81                  var listCE5 = gett5.Value;
82                  Estado idd5 = null;
83                  foreach (var item in listCE5)
84                  {
85                          idd5 = item;
86                          if (item.Id == data)
87                                  break;
88                  }
89                  var putt5 = await cont5.PutEstado(idd5.CodEstado,
                    ↪  idd5);
90                  break;
91          case nameof(ExtrasDaEscola):
92                  break;
93          case nameof(MantenedoraDaEscola):
94                  MantenedorasDasEscolasController cont6 = new
                    ↪  MantenedorasDasEscolasController();
95                  var gett6 = await cont6.GetMantenedoraDaEscola();
96                  var listCE6 = gett6.Value;
97                  MantenedoraDaEscola idd6 = null;
```

```
98                  foreach (var item in listCE6)
99                  {
100                         idd6 = item;
101                         if (item.Id == data)
102                                 break;
103                 }
104                 var putt6 = await
                    ↪ cont6.PutMantenedoraDaEscola(idd6.CodEntidade,
                    ↪ idd6);
105                 break;
106         case nameof(Municipio):
107                 MunicipiosController cont7 = new
                    ↪ MunicipiosController();
108                 var gett7 = await cont7.GetMunicipio();
109                 var listCE7 = gett7.Value;
110                 Municipio idd7 = null;
111                 foreach (var item in listCE7)
112                 {
113                         idd7 = item;
114                         if (item.Id == data)
115                                 break;
116                 }
117                 var putt7 = await
                    ↪ cont7.PutMunicipio(idd7.CodMunicipio, idd7);
118                 break;
```

```
119                 case nameof(Regiao):
120                         RegioesController cont8 = new RegioesController();
121                         var gett8 = await cont8.GetRegiao();
122                         var listCE8 = gett8.Value;
123                         Regiao idd8 = null;
124                         foreach (var item in listCE8)
125                         {
126                                 idd8 = item;
127                                 if (item.Id == data)
128                                         break;
129                         }
130                         var putt8 = await cont8.PutRegiao(idd8.CodRegiao,
                            ↪ idd8);
131                         break;
132                 case nameof(Telefone):
133                         TelefonesController cont9 = new
                            ↪ TelefonesController();
134                         var gett9 = await cont9.GetTelefone();
135                         var listCE9 = gett9.Value;
136                         Telefone idd9 = null;
137                         foreach (var item in listCE9)
138                         {
139                                 idd9 = item;
140                                 if (item.Id == data)
141                                         break;
```

```
142                                                     }
143                                                     var putt9 = await cont9.PutTelefone(idd9.Numero,
                                                        ↪ idd9.Ano, idd9.CodEntidade, idd9);
144                                                     break;
145                                             default:
146                                                     break;
147                                     }
148                             }
149                     }
150             }
151     }
152
153     public async Task DataDeletedEventHandler(Dictionary<uint, HashSet<ulong>> dbAndTables,
        ↪ Dictionary<ulong, HashSet<BigInteger>> tableAndData)
154     {
155             foreach (var db in dbAndTables)
156             {
157                     foreach (var table in db.Value)
158                     {
159                             var t = Globals.TablesTypes[table].Name;
160                             foreach (var data in tableAndData.GetValueOrDefault(table))
161                             {
162                                     switch (t)
163                                     {
164                                             case nameof(CensoEscola):
```

```
165                 CensoEscolasController cont = new
                    ↪ CensoEscolasController();
166                 var gett = await cont.GetCensoEscola();
167                 var listCE = gett.Value;
168                 CensoEscola idd = null;
169                 foreach (var item in listCE)
170                 {
171                         idd = item;
172                         if (item.Id == data)
173                                 break;
174                 }
175                 var putt = await cont.DeleteCensoEscola(idd.Ano,
                    ↪ idd.CodEntidade);
176                 break;
177         case nameof(CorreioEletronico):
178                 CorreioEletronicoController cont2 = new
                    ↪ CorreioEletronicoController();
179                 var gett2 = await cont2.GetCorreioEletronico();
180                 var listCE2 = gett2.Value;
181                 CorreioEletronico idd2 = null;
182                 foreach (var item in listCE2)
183                 {
184                         idd2 = item;
185                         if (item.Id == data)
186                                 break;
```

```
187                 }
188                 var putt2 = await
                    ↪  cont2.DeleteCorreioEletronico(idd2.Ano,
                    ↪  idd2.CodEntidade);
189                 break;
190         case nameof(Endereco):
191                 EnderecosController cont3 = new
                    ↪  EnderecosController();
192                 var gett3 = await cont3.GetEndereco();
193                 var listCE3 = gett3.Value;
194                 Endereco idd3 = null;
195                 foreach (var item in listCE3)
196                 {
197                         idd3 = item;
198                         if (item.Id == data)
199                                 break;
200                 }
201                 var putt3 = await
                    ↪  cont3.DeleteEndereco(idd3.CodEndereco);
202                 break;
203         case nameof(Escola):
204                 EscolasController cont4 = new EscolasController();
205                 var gett4 = await cont4.GetEscola();
206                 var listCE4 = gett4.Value;
207                 Escola idd4 = null;
```

```
208                 foreach (var item in listCE4)
209                 {
210                         idd4 = item;
211                         if (item.Id == data)
212                                 break;
213                 }
214                 var putt4 = await
                    ↪  cont4.DeleteEscola(idd4.CodEntidade);
215                 break;
216         case nameof(Estado):
217                 EstadosController cont5 = new EstadosController();
218                 var gett5 = await cont5.GetEstado();
219                 var listCE5 = gett5.Value;
220                 Estado idd5 = null;
221                 foreach (var item in listCE5)
222                 {
223                         idd5 = item;
224                         if (item.Id == data)
225                                 break;
226                 }
227                 var putt5 = await
                    ↪  cont5.DeleteEstado(idd5.CodEstado);
228                 break;
229         case nameof(ExtrasDaEscola):
230                 MongoController cont10 = new MongoController();
```

```
231                 var gett10 = await cont10.GetExtraAsync();
232                 var listCE10 = gett10.Value;
233                 ExtrasDaEscola idd10 = null;
234                 foreach (var item in listCE10)
235                 {
236                         idd10 = item;
237                         if (item.Id == data)
238                                 break;
239                 }
240                 var putt10 = await cont10.DeleteExtra(idd10.ID.Ano,
                    ↪ idd10.ID.Cod_Entidade);
241                 break;
242         case nameof(MantenedoraDaEscola):
243                 MantenedorasDasEscolasController cont6 = new
                    ↪ MantenedorasDasEscolasController();
244                 var gett6 = await cont6.GetMantenedoraDaEscola();
245                 var listCE6 = gett6.Value;
246                 MantenedoraDaEscola idd6 = null;
247                 foreach (var item in listCE6)
248                 {
249                         idd6 = item;
250                         if (item.Id == data)
251                                 break;
252                 }
```

```
253                 var putt6 = await
                    ↪  cont6.DeleteMantenedoraDaEscola(idd6.CodEntidade);
254                 break;
255         case nameof(Municipio):
256                 MunicipiosController cont7 = new
                    ↪  MunicipiosController();
257                 var gett7 = await cont7.GetMunicipio();
258                 var listCE7 = gett7.Value;
259                 Municipio idd7 = null;
260                 foreach (var item in listCE7)
261                 {
262                         idd7 = item;
263                         if (item.Id == data)
264                                 break;
265                 }
266                 var putt7 = await
                    ↪  cont7.DeleteMunicipio(idd7.CodMunicipio);
267                 break;
268         case nameof(Regiao):
269                 RegioesController cont8 = new RegioesController();
270                 var gett8 = await cont8.GetRegiao();
271                 var listCE8 = gett8.Value;
272                 Regiao idd8 = null;
273                 foreach (var item in listCE8)
274                 {
```

```
275                     idd8 = item;
276                     if (item.Id == data)
277                         break;
278                 }
279                 var putt8 = await
                    ↪ cont8.DeleteRegiao(idd8.CodRegiao);
280                 break;
281             case nameof(Telefone):
282                 TelefonesController cont9 = new
                    ↪ TelefonesController();
283                 var gett9 = await cont9.GetTelefone();
284                 var listCE9 = gett9.Value;
285                 Telefone idd9 = null;
286                 foreach (var item in listCE9)
287                 {
288                     idd9 = item;
289                     if (item.Id == data)
290                         break;
291                 }
292                 var putt9 = await cont9.DeleteTelefone(idd9.Numero,
                    ↪ idd9.Ano, idd9.CodEntidade);
293                 break;
294             default:
295                 break;
296         }
```

297
298
299
300
301
302

APÊNDICE B – GLOBAIS AUXILIARES

```
5  namespace API.Comunicacao
6  {
7          public static class Globals
8          {
9                  public static Dictionary<ulong, Type> TablesTypes = new
                   ↪  Dictionary<ulong, Type>();
10                 public static HashSet<(uint, string)> Dbs = new
                   ↪  HashSet<(uint, string)>();
11                 public static HashSet<ulong> Tables = new
                   ↪  HashSet<ulong>();
12
13                 private static readonly Dictionary<uint, HashSet<ulong>>
                   ↪  pdbAndTables = new Dictionary<uint,
                   ↪  HashSet<ulong>>();
14
15                 public static Dictionary<uint, HashSet<ulong>>
                   ↪  DbAndTables
16                 {
17                         get { return pdbAndTables; }
18                 }
19
20                 private static readonly Dictionary<ulong,
                   ↪  HashSet<BigInteger>> ptableAndData = new
                   ↪  Dictionary<ulong, HashSet<BigInteger>>();
21
22                 public static Dictionary<ulong, HashSet<BigInteger>>
                   ↪  TableAndData
23                 {
24                         get { return ptableAndData; }
25                 }
26
27                 public static void AddTable(ulong tId, Type ty)
28                 {
29                         Tables.Add(tId);
30                         TablesTypes.Add(tId,ty);
31                         ptableAndData.Add(tId, new
                           ↪  HashSet<BigInteger>());
```

```
32                  }
33                  public static void AddDB(uint dbId, string uri)
34                  {
35                          Dbs.Add((dbId, uri));
36                          pdbAndTables.Add(dbId, new HashSet<ulong>());
37                  }
38
39                  public static void AssocDbAndTable(uint dbId, ulong tId)
40                  {
41                          pdbAndTables[dbId].Add(tId);
42                  }
43
44                  public static void AssocTableAndData(ulong tId,
                    ↪  BigInteger dId)
45                  {
46                          ptableAndData[tId].Add(dId);
47                  }
48
49          }
50  }
```

APÊNDICE C – CONTROLADOR COMUNICAÇÃO

```
5  using System.Net.Http;
6  using System.Threading.Tasks;
7
8  namespace API.Comm.Controllers
9  {
10         [Route("api/Comm/[controller]")]
11         [ApiController]
12         public class CensoEscolasController : ControllerBase
13         {
14                 private string uri;
15                 public CensoEscolasController(string u = "localhost/api/CensoEscolas")
16                 {
17                         uri = u;
18                 }
19
20                 // GET: api/CensoEscolas
21                 [HttpGet]
22                 public async Task<ActionResult<IEnumerable<CensoEscola>>> GetCensoEscola()
23                 {
24                         using (HttpClient client = new HttpClient())
25                         {
26                                 ActionResult<IEnumerable<CensoEscola>> ret = null;
27                                 var response = await client.GetAsync(uri);
```

```
28
29                      if (response.IsSuccessStatusCode)
30                      {
31                              var t = await response.Content.ReadAsStringAsync();
32                              ret =
                                ↪  JsonConvert.DeserializeObject<ActionResult<IEnumerable<CensoEscola>>>(t);
33                      }
34                      return ret;
35              }
36      }
37
38      // GET: api/CensoEscolas/2018/101010
39      [HttpGet("{ano}/{id}")]
40      public async Task<ActionResult<CensoEscola>> GetCensoEscola(short ano, long id)
41      {
42              using (HttpClient client = new HttpClient())
43              {
44                      ActionResult<CensoEscola> ret = null;
45                      var response = await client.GetAsync(uri + "/" + ano + "/" + id);
46
47                      if (response.IsSuccessStatusCode)
48                      {
49                              var t = await response.Content.ReadAsStringAsync();
50                              ret = JsonConvert.DeserializeObject<ActionResult<CensoEscola>>(t);
51                      }
```

```
52                  return ret;
53          }
54  }
55
56  // PUT: api/CensoEscolas/2018/101010
57  // To protect from overposting attacks, please enable the specific properties you want to bind to,
    ↪  for
58  // more details see https://aka.ms/RazorPagesCRUD.
59  [HttpPut("{ano}/{id}")]
60  public async Task<IActionResult> PutCensoEscola(short ano, long id, CensoEscola censoEscola)
61  {
62          using (HttpClient client = new HttpClient())
63          {
64                  IActionResult ret = null;
65                  StringContent cont = new StringContent(JsonConvert.SerializeObject(censoEscola));
66                  var response = await client.PutAsync(uri + "/" + ano + "/" + id, cont);
67
68                  if (response.IsSuccessStatusCode)
69                  {
70                          var t = await response.Content.ReadAsStringAsync();
71                          ret = JsonConvert.DeserializeObject<IActionResult>(t);
72                  }
73                  return ret;
74          }
75  }
```

```
76
77          // POST: api/CensoEscolas
78          // To protect from overposting attacks, please enable the specific properties you want to bind to,
            ↪  for
79          // more details see https://aka.ms/RazorPagesCRUD.
80          [HttpPost]
81          public async Task<ActionResult<CensoEscola>> PostCensoEscola(CensoEscola censoEscola)
82          {
83                  using (HttpClient client = new HttpClient())
84                  {
85                          ActionResult<CensoEscola> ret = null;
86                          StringContent cont = new StringContent(JsonConvert.SerializeObject(censoEscola));
87                          var response = await client.PostAsync(uri + "/", cont);
88
89                          if (response.IsSuccessStatusCode)
90                          {
91                                  var t = await response.Content.ReadAsStringAsync();
92                                  ret = JsonConvert.DeserializeObject<ActionResult<CensoEscola>>(t);
93                          }
94                          return ret;
95                  }
96          }
97
98          // DELETE: api/CensoEscolas/2019/101010
99          [HttpDelete("{ano}/{id}")]
```

```
100             public async Task<ActionResult<CensoEscola>> DeleteCensoEscola(short ano, long id)
101             {
102                     using (HttpClient client = new HttpClient())
103                     {
104                             ActionResult<CensoEscola> ret = null;
105                             var response = await client.DeleteAsync(uri + "/" + ano + "/" + id);
106 
107                             if (response.IsSuccessStatusCode)
108                             {
109                                     var t = await response.Content.ReadAsStringAsync();
110                                     ret = JsonConvert.DeserializeObject<ActionResult<CensoEscola>>(t);
111                             }
112                             return ret;
113                     }
114             }
115     }
116 }
```

Sistema de Gerência de Bancos de Dados baseado em Blockchain

Vinicius S. Berkenbrock[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brazil

vinaosb@grad.ufsc.br

Abstract. *The goal of the work is to implement a Blockchain to prevent distributed databases from owning a single point of failure, named "manager", avoiding problems such as falling network of part of banks, ensuring integrity and increasing the security of past data between databases beyond enabling the use of different database technologies. The chosen method is the use of a Blockchain to ensure that there is consensus among all nodes of the network, thus avoiding the errors cited above at the same time as no central nod is required for data management as well as adding a layer of data provenance.*

Resumo. *O objetivo deste trabalho é implementar uma Blockchain para evitar com que os bancos de dados distribuídos possuam um ponto único de falha, denominado "gerente de bancos", evitando problemas como queda de rede de parte dos bancos, garantir a integridade e aumentar a segurança dos dados passados entre os bancos de dados além de possibilitar o uso de diferentes tecnologias de bancos de dados. O método escolhido é o uso de uma Blockchain para garantir que haja consenso entre todos os nodos da rede, evitando assim os erros citados acima ao mesmo tempo pois não é necessário um nodo central para a gerência dos dados assim como adicionando uma camada de proveniência de dados.*

1. Contextualização

Para contextualizar a implementação, tomaremos o cenário atual de como são armazenados os dados gerados diariamente na maior parte das empresas. Dados geralmente são apresentados geralmente de duas formas distintas: apenas Dados ou *Big Data*. A diferença entre eles é que geralmente *Big Data* muitas vezes possui um volume muito grande de dados que em sua maioria não são relevantes para quem os armazena, mas que tais dados que seriam considerados inúteis, possam ser usados em pesquisa ou para geração de novos dados a partir do que já é sabido.

Para o armazenamento desses dados hoje são empregados soluções de Bancos de Dados muito solidificadas no mercado, como Bancos de Dados SQL. Que em sua maioria são Bancos de Dados pensados para trabalhar de forma centralizada e que nem sempre são compatíveis com Bancos de Dados concorrentes.

Houve também uma adoção de mercado de Bancos de Dados NoSQL recentemente por trabalharem melhor de forma descentralizada por não oferecer certas garantias que os Bancos de Dados SQL oferecem.

Contudo essas tecnologias ainda tratam a descentralização dos dados armazenados usando métodos de eleição; Esses métodos fazem com que sempre um dos Bancos de Dados fique responsável de transmitir dados para os demais.

Solucionando esse problema surgiram as tecnologias de *Blockchain*, que usam de redes Peer-to-Peer para transmitir informações e armazenam dados por replicação, sendo que todos os nodos de uma rede teriam o mesmo conteúdo. Isso faz com que haja por muitas vezes, replicação excessiva de dados. Eles não usam de algoritmos de eleição de responsáveis mas trabalham de forma completamente diferente.

A poucos anos, surgiu a ideia de *Smart-Contracts* que são basicamente algoritmos de computador que rodam em nodos de uma rede *Blockchain*. Usando dessa tecnologia podemos então realizar tarefas mais complexas do que apenas armazenar dados.

Para exemplificar a implementação, tomaremos os bancos de dados usados pelo governo brasileiro, e fornecidos no Portal Brasileiro de Dados Abertos [do Brasil]. Eles são fornecidos em diversos formatos, muitos deles com pouca adoção de mercado, o que os torna muito difíceis de serem mantidos e de se realizar tarefas de grande valor além do que já havia sido previsto, assim limitando o seu potencial.

Temos aqui porém um problema, a base de dados brasileira é de tamanho substancial, algo que poderia tornar o uso de uma *Blockchain* inviável para o armazenamento desses dados.

Para solucionar esse problema, esse trabalho une as tecnologias de Bancos de Dados existentes com um controle baseado em *Blockchain*, para que não acabe ocorrendo redundância excessiva de dados.

2. Definição do Problema

No setor atual de tecnologia, surgiu a tecnologia chamada *Blockchain* a qual está tendo muito pouca adoção na indústria já consolidada, mas ao mesmo tempo sendo muito popular com *startups*, como apresentado por [Due 2019]. O motivo está justamente nas desvantagens da implantação de uma Blockchain a longo prazo: o alto consumo da capacidade física espacial dos servidores usados.

Atualmente são usados tecnologias de Bancos de Dados para solucionar todos os tipos de problemas relacionados à armazenamento de dados que poderiam ser solucionados em conjunto com *Blockchains*. Usam-se hoje, Sistemas de Gerência de Bancos de Dados disponíveis no mercado atual, que em sua maioria foram criados com a mentalidade de centralização, e não distribuição (como o uma *Blockchain*), de poder computacional. Alguns Sistemas de Gerência de Bancos de Dados Distribuídos (como PostgreSQL) também foram criados com o intuito de distribuir tais arquiteturas, mas ainda são inerentemente centralizados, requisitando um nodo responsável para poder realizar suas operações (Vide documentação: [PostgreSQL]).

Outro problema está na organização da proveniência dos dados, pois os Bancos de Dados podem não possuir ferramentas o suficiente por si mesmos para garantir a proveniência de dados criados e/ou alterados, como dados bancários ou que possuam dados pessoais dos usuários do sistema que podem resultar em problemas graves caso estejam errados ou sejam inseridos por pessoas não autorizadas como por exemplo, emissão de passaportes que devem ser realizadas apenas por autoridades e devem ser rastreáveis,

podendo assim ser explorados por usuários hackers, sendo muitas vezes necessário a reversão do Bancos de Dados para o último estado válido, e nem sempre com garantia total de que esse estado escolhido não havia sido previamente atacado.

3. Busca do Estado da Arte

A busca por trabalhos na área de pesquisa foi realizada em dois momentos, nos dias 10/02/2019 e 20/10/2019. A tabela 1 exibe a quantidade de artigos encontrados durante a revisão do estado da arte. Vale salientar que foram feitos refinamentos e testes com as *strings* de buscas para que este resultado fosse obtido.

Como é possível de se perceber, a tabela é dividida em 3 etapas:

1. Busca Inicial: o número de artigos presentes nesta coluna corresponde ao número total de artigos encontrados.
2. Filtro 1: o número de artigos presentes nesta coluna foram os que tiveram títulos e resumos lidos e tiveram alguma relevância com o tema desse projeto.
3. Filtro 2: exibe o número de artigos que foram lidos por completo, selecionados da etapa anterior.

Table 1. Numero de artigos encontrados durante revisão literária.

Repositório	Busca Inicial	Filtro 1	Filtro 2
IEEE Xplore Digital Library	21	5	3
Google Scholar	30	0	0
ACM Digital Library	59	8	5
Total	110	13	8

À partir da busca, foram encontrados 110 artigos ao todo, sendo que apenas 8 foram de extrema importância e passaram o segundo filtro apresentado na tabela 1. Como esta é uma área relativamente nova na computação em geral, houve dificuldade na busca de artigos que abordassem a temática "*Blockchain* como coordenador de tarefas", sendo tarefa qualquer assunto que possa ser coordenado, como procedimentos de algoritmos distribuídos ou transações de bancos de dados, principalmente porque as aplicações e sistemas existentes focam apenas na parte de armazenamento da proveniência dos dados na *Blockchain*.

4. Discussão de Resultados das Buscas

Detalhando a tabela acima, a coluna *Blockchain* deve-se o uso de uma *Blockchain* ou *Smart-Contract* na implementação da solução provida pelo artigo; a coluna *Proveniência de Dados* implica que foi explorado o uso da *Blockchain* para garantir um sistema de Proveniência de Dados; a coluna *Blockchain guarda meta-dados de um Banco de Dados* implica no uso de uma *Blockchain* para guardar meta-dados de um Banco de Dados ao invés de guardar os dados em si economizando assim espaço na *Blockchain* em si; a coluna *Blockchain Coordena um Banco de Dados* implica que a solução utilizada faz com que a *Blockchain* coordene um sistema de Banco de Dados; a coluna *Sharding* implica no uso de técnicas de *Sharding* ou exploração dessas técnicas para a solução desenvolvida.

Table 2. Comparação entre artigos e assuntos abordados.

Artigos	Blockchain	Proveniência de Dados	Blockchain guarda meta-dados de um BD	Blockchain Coordena um BD	Sharding
[Keller and Kessler 2018]	x	x			
[LIANG et al. 2017]	x	x	x		
[Cui et al. 2019]	x	x	x		
[Ramachandran and Ramachandran 2018]	x	x	x		
[Ezhilchelvan et al. 2018]	x		x	x	
[Dang et al. 2019]	x				x
[Bragagnolo et al. 2018]	x				
[Bragagnolo et al. 2019]	x				
Esta Proposta	x	x	x	x	x

É possível verificar na tabela 2 que os trabalhos encontrados não focam em todos os aspectos relevantes do escopo dessa proposta na sua maioria por não forçar com que o Banco de Dados usado para armazenamento de dados tenha o seu acesso depois de ter um comando da *Blockchain*.

Como pode ser observado, mesmo os trabalhos mais completos focados em mostrar um sistema em Blockchain que guarda meta-dados de um Banco de Dados acabam focando mais na proveniência dos dados do que no controle e *sharding* do Banco de Dados.

5. Modelagem da API

O paradigma das *APIs* utilizado é o paradigma *REST*. As *APIs* serão divididas em 3 categorias como descrito acima, uma *API* de comunicação, onde irá realizar as operações que o usuário enviar e irá coordenar as demais *APIs*; uma *API* para comunicação com a *Blockchain*, que se tornará responsável de armazenar e consultar metadados armazenados no *Smart-Contract*; e duas *APIs* de comunicação com dois diferentes Bancos de Dados.

A Figura 1 apresenta a modelagem conceitual da intercomunicação das *API* desenvolvidas. Vale lembrar que a *API* usada é do tipo *REST*.

6. Blockchain

A *Blockchain* escolhida para a realização deste trabalho é a *Ethereum*, a qual fornece suporte à *Smart-Contract* ao mesmo tempo sendo a *Blockchain* mais usada para esse fim.

O *Smart-Contract* foi feito usando a linguagem Solidity, que é a mais usada, e a com maior suporte da comunidade, tendo em vista que as demais ou possuem uma falta de suporte ou estão depreciadas.

O *Smart-Contract* deverá ter como meta-dados:

- Meta-dados dos Dados Global:
  - Hash do Dado.
  - Status do Dado (se foi deletado ou não).
- Meta-dados das Tabelas:
  - Mapeamento dos meta-dados dos dados que da Tabela.

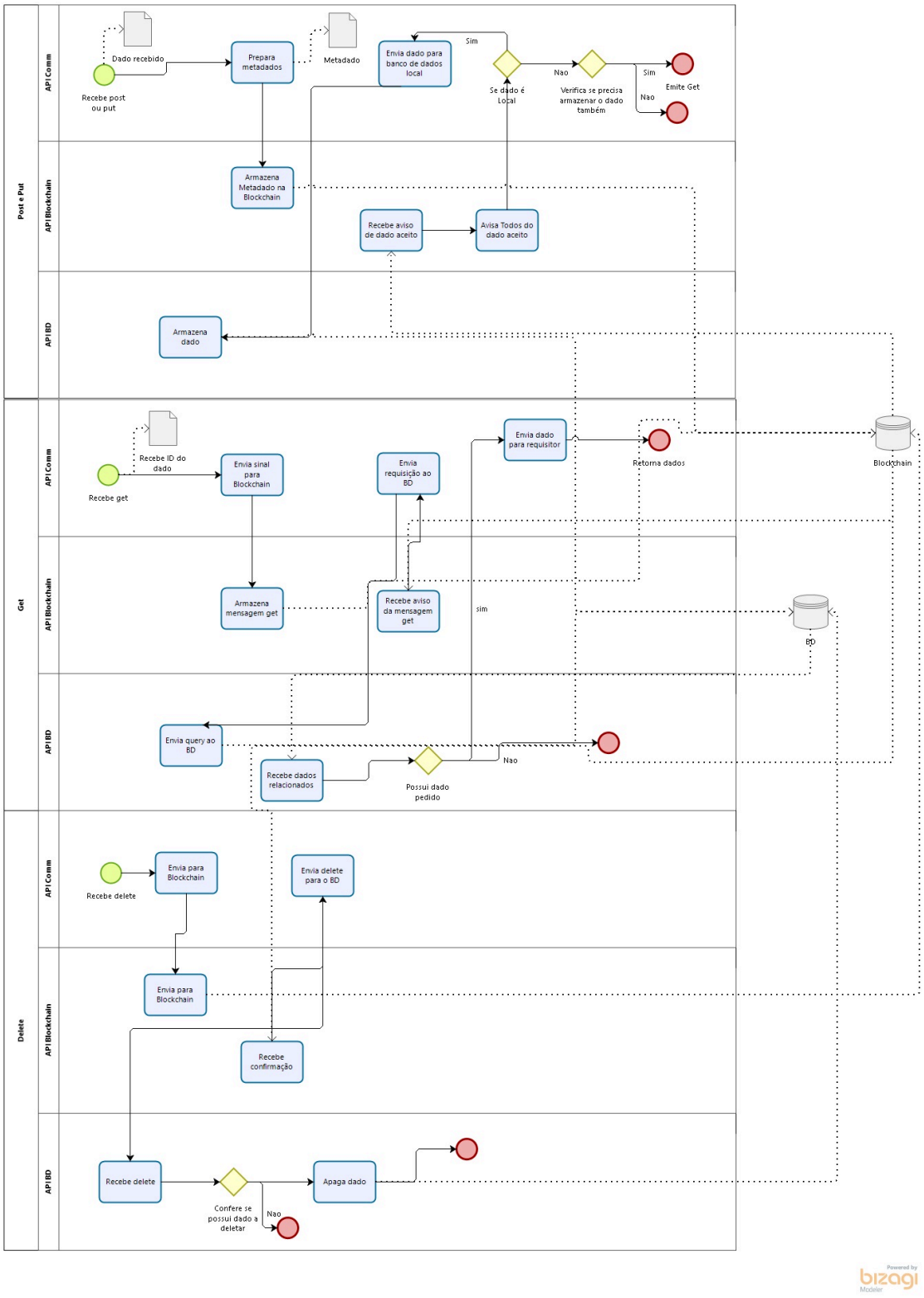


Figure 1. Modelagem do funcionamento das *APIs*.

  - Lista dos meta-dados dos atributos dos dados da Tabela.
  - Numero de dados de uma Tabela.
- Meta-dados dos BD:
  - Meta-dados de quais tabelas estão no BD.
  - Meta-dados de quais dados estão na tabela desse BD. (*Sharding* vertical).
  - Meta-dados de quais atributos dos dados estão na tabela. (*Sharding* horizontal).

Além disso o *Smart-Contract* deverá possuir eventos para alertar a *API* própria quando ocorrer alguma mudança para que ela avise a *API* de controle quais BD estão envolvidos nessa mudança e caso algum deles precise realizar ações. Esses eventos são:

- Evento de Criação de BD.
- Evento de Criação de Tabela com Codificação de Atributos e adição da mesma no BD. Isso faz com que haja possibilidade de replicação dos dados quando necessário e permite *Sharding*.
- Evento de Criação de Dado Globalmente.
- Evento de Adição de Dado inteiro ou parcial em uma Tabela.
- Evento de Modificação de Dado Globalmente.
- Evento de Deleção de Dado Globalmente.

No caso dos *Smart-Contracts* sempre que há um evento, também é registrado quem realizou aquele evento em um Log. Esse log pode ser acessado para realizar estudos de proveniência.

7. Bancos de Dados

As tecnologias para o desenvolvimento de BD, foram escolhidas pela familiaridade com suas ferramentas e boas características para uso no cenário. Segue abaixo os SGBD escolhidos:

- *BD SQL*: PostgreSQL.
- *BD NoSQL*: MongoDB.

Foi criado também em nível de *API*, uma abstração no conceito de BD, usando um *O/RM* conhecido por *EF Core*.

O esquema do BD foi primeiramente modelado, usando a *1FN*, nos dados retirados do [do Brasil]. Originalmente os dados fornecidos já estavam nesse formato. Em seguida foi aplicado a *2FN*, a qual pouco mudou o esquema. Termina-se a modelagem usando a *3FN* que mudou radicalmente o esquema.

Com a *3FN*, é possível gerar o Diagrama Entidade Relacional apresentado na Figura 2 e com base nela determinar quais atributos que ocorrem de estar ausentes em alguns dados, esses dados serão armazenados no *BD NoSQL*. Os atributos que estão sempre presentes, serão adicionados no *BD SQL*. Além disso foi decidido por retirar as tabelas ”Censo” e ”Dependência Administrativa” e adicionar seus atributos na tabela ”Censo_Escola” por gerarem complicações desnecessárias ao BD.

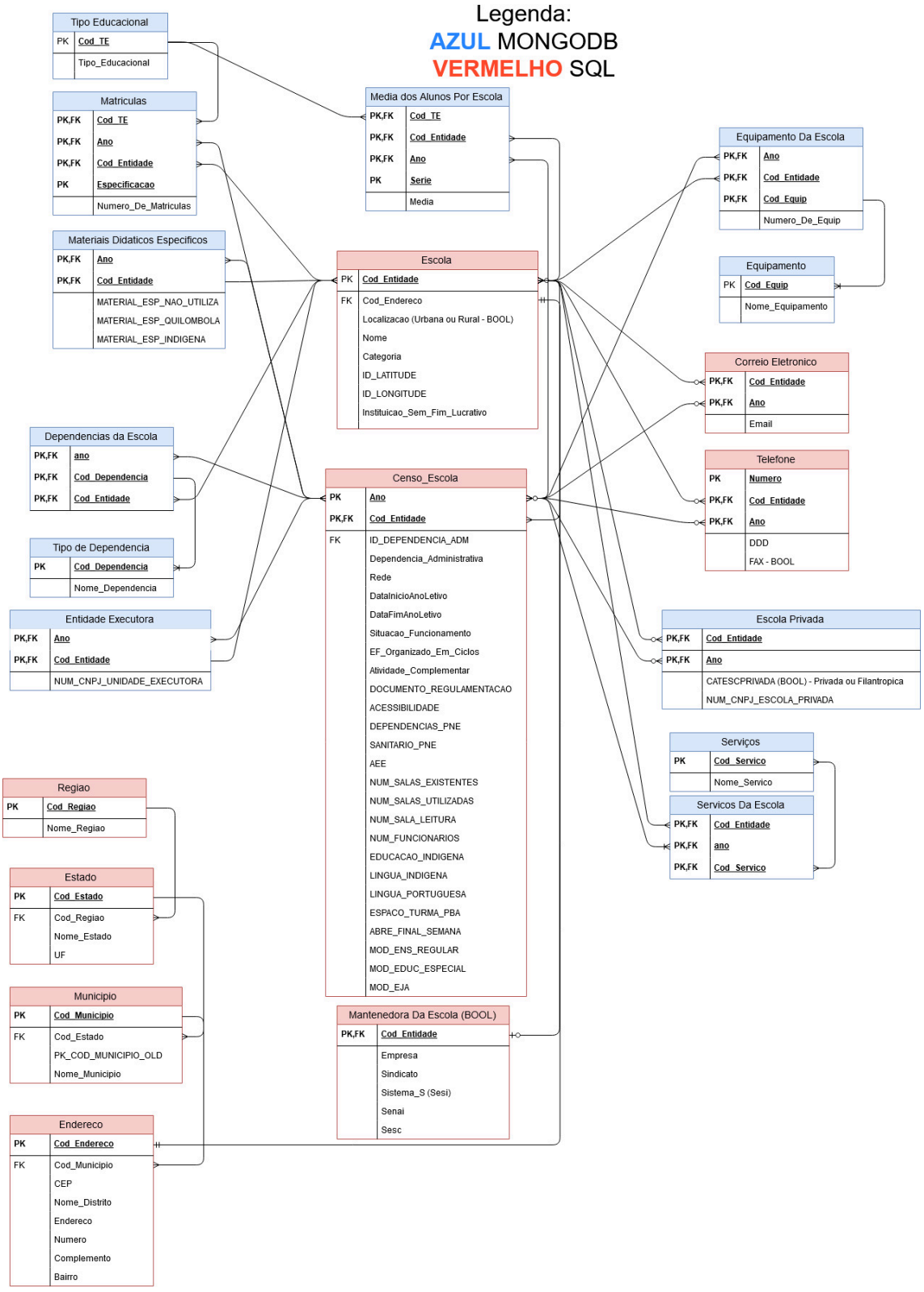


Figure 2. Diagrama Entidade Relacional

8. Conclusão

Esse trabalho conseguiu implementar uma arquitetura para ambientes que precisam garantir a veracidade e proveniência dos dados e que não se importam com os problemas de atraso acarretados da *Blockchain*, ao mesmo tempo que, realiza um uso eficiente da *Blockchain* ao armazenar apenas meta-dados dos dados armazenados nos Bancos de Dados.

9. Trabalhos Futuros

A seguir é provido uma lista com alguns temas de trabalhos futuros sugeridos:

- Substituição do Contrato Inteligente por uma *Blockchain* própria a fim de aumentar o desempenho e diminuir o tamanho da *Blockchain* em si e talvez aumentar a generalização proposta para que tenha mais propósitos.
- Ampliação do uso, criando novos cenários de aplicação.
- Criação de bibliotecas genéricas para rápido desenvolvimento de novos sistemas.
- Ao invés de a *Blockchain* coordenar diferentes Bancos de Dados, coordenar diferentes nodos em uma rede de processamento paralelo, assim diminuindo a dependência de um nodo mestre como ocorre atualmente na área de processamento paralelo de dados.
- Aplicar conhecimentos para utilização em nuvem e contêineres.

References

Bragagnolo, S., Rocha, H., Denker, M., and Ducasse, S. (2018). Ethereum query language.

Bragagnolo, S., Rocha, H., Denker, M., and Ducasse, S. (2019). Blockchaindb - towards a shared database on blockchains.

Cui, H., Chen, Z., Xi, Y., Chen, H., and Hao, J. (2019). Iot data management and lineage traceability: A blockchain-based solution.

Dang, H., Dinh, T. T. A., Loghin, D., Chang, E.-C., Lin, Q., and Ooi, B. C. (2019). Towards scaling blockchain systems via sharding.

do Brasil, R. F. Conjuntos de dados - portal brasileiro de dados abertos.

Due (2019). Blockchain adoption barriers in startups and enterprises.

Ezhilchelvan, P., Aldweesh, A., and Moorsel, A. v. (2018). Non-blocking two-phase commit using blockchain.

Keller, T. and Kessler, N. (2018). Yet another blockchain use case – the label chain.

LIANG, X., SHETTY, S., TOSH, D., KAMHOUA, C., KWIAT, K., and NJILLA, L. (2017). Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability.

PostgreSQL. Documentation: 12: Chapter 26, high availability, load balancing and replication.

Ramachandran, A. and Ramachandran, A. (2018). Smartprovenance: A distributed, blockchain based dataprovenance system.