UNIVERSIDADE FEDERAL DE SANTA CATARINA

Uma Ferramenta para Extração de Esquemas de Bancos de Dados NoSQL Orientados a Documentos

Felipe de Souza da Costa

Florianópolis

2017/2

Felipe de Souza da Costa

Uma Ferramenta para Extração de Esquemas de Bancos de Dados NoSQL Orientados a Documentos

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do título de Bacharel, do curso de Sistemas de Informação na Universidade Federal de Santa Catarina.
Orientador: Profº Ronaldo dos Santos Mello, Dr.
Coorientador: Profº Angelo Augusto Frozza, M.Sc.

Florianópolis

2017/2

Felipe de Souza da Costa

Uma Ferramenta para Extração de Esquemas de Bancos de Dados NoSQL Orientados a Documentos

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do título de Bacharel, do curso de Sistemas de Informação na Universidade Federal de Santa Catarina.

Profº Cristian Koliver, Dr.
Coordenador do Curso

Banca Examinadora:

Profº Ronaldo dos Santos Mello, Dr.
Orientador
Universidade Federal de Santa Catarina

Profº Angelo Augusto Frozza, M.Sc.
Coorientador
Instituto Federal Catarinense

Profª Carina Friedrich Dorneles, Drª
Universidade Federal de Santa Catarina

Profº Roberto Willrich, Dr.
Universidade Federal de Santa Catarina

Florianópolis
2017/2

Dedico este trabalho a todas as pessoas que, de algum modo, me incentivaram antes e durante essa etapa tão importante em minha vida.

Agradecimentos

Agradeço primeiramente à minha família, em especial aos meus pais, por terem sempre fornecido a melhor educação ao seu alcance. A minha esposa, que incessavelmente me apoiou antes e durante esta longa caminhada e que mesmo em momentos difíceis, sempre esteve ao meu lado me apoiando, motivando e incentivando. Ao meu sogro e minha sogra, por vibrarem comigo a conquista de passar no vestibular e por concluir essa importante etapa em minha vida.

A todo o corpo docente do CTC-UFSC, pelos ensinamentos que em alguns casos fogem às fronteiras do técnico e do cientifico. Em especial aos meus orientadores, Profº Ronaldo e Profº Angelo, pelos conhecimentos, ideias e incentivo durante o desenvolvimento desse trabalho e por estarem sempre a disposição quando eu precisava de ajuda. Aos professores da banca, Profª Carina e Profº Roberto, por aceitarem o convite.

Aos meus amigos, pelo companheirismo, pelas noites em claro fazendo trabalhos via Skype, pelas gargalhadas, pelos conselhos e pela paciência durante esta caminhada.

"Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar a onde a maioria não chega, faça o que a maioria não faz."

(Bill Gates)

Resumo

Os bancos de dados NoSQL têm se tornando cada vez mais populares no desenvolvimento de aplicações, entre outras características, devido à sua capacidade de lidar com grandes volumes de dados e pela ausência de um esquema de dados explícito. Embora a maioria dos bancos de dados NoSQL não tenha esquema, as informações sobre as propriedades estruturais dos dados persistidos são essenciais durante o desenvolvimento de aplicações. Sem o conhecimento dessas propriedades estruturais, atividades de desenvolvimento de aplicações ou análise de dados tornam-se um trabalho custoso e, às vezes, impraticáveis. Sendo assim, o presente trabalho propõe o desenvolvimento de uma ferramenta que extraia o esquema de uma coleção de documentos no formato JSON, armazenados em um banco de dados NoSQL orientado a documentos, com o objetivo de facilitar diversas tarefas de manipulação posterior desses dados, como a recuperação, validação, integração e análise de dados. Na fase de extração das estruturas dos documentos são aplicadas operações de agregação visando obter um documento para cada estrutura distinta e também é proposta uma estrutura global para agrupar essas estruturas a fim de gerar um único esquema no formato JSON *Schema*. Finalmente, experimentos realizados em *datasets* reais do *DBPedia*, *Foursquare* e *GitHub*, além de um *dataset* hipotético, demonstram que os resultados de tempo de processamento e a completude dos esquemas gerados são comparáveis com os resultados encontrados em abordagens do estado da arte.

Palavras-chave: Extração de esquemas, NoSQL, Banco de Dados orientado a documentos, JSON, JSON Schema.

Abstract

NoSQL databases are becoming increasingly popular in application development, among other features, because of their ability to handle large data volumes and also their ability to be *schemaless*. Although most NoSQL databases are schemaless, information about the structural properties of stored data is essential during the application development. Without the knowledge of these structural properties, application development or data analysis activities become costly and sometimes unfeasible. Thus, this work proposes a tool that, given a collection of data in JSON format, stored in a document-oriented NoSQL database, performs the extraction of its schema, with the purpose of facilitating further data manipulation tasks, like data retrieval, integration, validation and analysis. In the extraction phase of the document structure, aggregation operations are applied to obtain a document for each distinct structure. Besides, a global structure is proposed to group these structures in order to present a single schema in JSON *Schema* format. Finally, experiments based on real *DBPedia*, *Foursquare* and *GitHub* datasets, as well as a hypothetical dataset, demonstrate that the results of processing time and completeness of the schemes generated are comparable with the results found in state of the art approaches.

Keywords: Schema Extraction, NoSQL, Document-oriented Database, JSON, JSON Schema.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

ACID	*Atomicity*, *Consistency*, *Isolation*, *Durability*
API	*Application Programming Interface*
BASE	*Basically Available*, *Soft state*, *Eventual consistency*
BD	Banco de Dados
BLOB	*Binary Large OBject*
BSON	*Binary* JSON
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
DTD	*Document Type Definition*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
MDE	*Model Driven Engineering*
MVC	*Model, View, Controller*
NoSQL	*Not* SQL ou *Not Only* SQL
OWL	*Web Ontology Language*
SGBD	Sistema de Gerenciamento de Banco de Dados
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RF	Requisito Funcional
RNF	Requisito Não-Funcional
TCC	Trabalho de Conclusão de Curso

UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*
WWW	*World Wide Web*

Sumário

1 Introdução

A explosão do mercado de *Big Data* fez com que grandes empresas demandassem por bancos de dados (BDs) capazes de armazenar e processar grandes volumes de dados de forma eficaz, com um alto desempenho em operações de leitura e escrita. Sendo assim, os BDs relacionais começaram a enfrentar vários desafios quanto a alta concorrência em operações de leitura e escrita, armazenamento de *Big Data* de forma eficiente, capacidade de suportar expansões horizontais e *upgrades*, garantir um serviço rápido e ininterrupto, além de ter menores custos de gestão e de operação (HAN et al., 2011).

Para atender essas necessidades, uma variedade de sistemas de BDs surgiu, com o propósito de facilitar a expansão, apoiar o armazenamento em massa, suportar eficientemente operações de leitura e escrita e, além de tudo, ter um baixo custo de operação e manutenção. Para isso, esses BDs dispensam as propriedades ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*) dos BDs relacionais em troca de manter as propriedades BASE (*Basically Available*, *Soft state*, *Eventual consistency*) (TUDORICA; BUCUR, 2011). A esses BDs é dado o nome de BD NoSQL — conceito definido em 1998 por Carlo Strozzi e reintroduzido em 2009, por Eric Evans, em um evento realizado em São Francisco (EUA), sobre BDs não relacionais, distribuídos e de código aberto (SADALAGE; FOWLER, 2012).

O termo NoSQL é um acrônimo para "*Not Only* SQL" ou "*Not* SQL", uma definição que não está totalmente acordada no meio acadêmico (CATTELL, 2011; SADALAGE; FOWLER, 2012; RUIZ; MORALES; MOLINA, 2015). No contexto deste trabalho, o termo NoSQL é utilizado no sentido de "*Not* SQL", pois interpretar um BD como "*Not Only* SQL" pode tornar o termo sem sentido, já que um BD NoSQL não é de fato um BD SQL. O termo "*Not Only* SQL" faz mais sentido ao se considerar o ecossistema de BDs, porém, para isso já existe outra *buzzword*[1]: Persistência Poliglota (SADALAGE; FOWLER, 2012).

Uma característica muito importante de BDs NoSQL é o fato de serem *schemaless*, ou seja, permitirem o armazenamento de dados sem o conhecimento prévio da estrutura dos mesmos (SADALAGE; FOWLER, 2012; RUIZ; MORALES; MOLINA,

[1] *Buzzeword*: uma palavra ou frase que está na moda em um momento específico ou em um contexto particular.

2015).

O modelo de dados de BDs tradicionais é, principalmente, o modelo relacional, suportando operações de associação de tabelas e transações ACID. No entanto, em NoSQL, não há um modelo de dados padrão a ser utilizado. Cada solução adota um modelo de dados diferente que podem ser classificados em quatro categorias amplamente utilizadas: chave-valor, orientado a documento, orientado a colunas e orientado a grafos.

Os BDs NoSQL orientados a documentos, por exemplo, armazenam e recuperam documentos, os quais podem ser nos formatos XML[2] (*eXtensible Markup Language*), JSON[3] (*JavaScript Object Notation*) (ver seção 2.4) ou BSON[4] (*Binary* JSON) (ver seção 2.4.1) (HASHEM; RANC, 2016; INDRAWAN-SANTIAGO, 2012; HAN et al., 2011). Em geral, não possuem esquema, ou seja, os documentos não precisam possuir uma estrutura em comum. De acordo com *DB-Engines Ranking*[5] os BDs mais populares dessa categoria são: *MongoDB*[6] (ver seção 2.3.5.1), *Amazon DynamoDB*[7], *Couchbase*[8] e *CouchDB*[9].

Em BDs relacionais, antes de armazenar dados é preciso ter uma estrutura previamente definida — que indica quais tabelas e colunas existem e quais tipos de dados cada coluna pode armazenar — ou seja, um esquema. No contexto NoSQL, armazenar dados é muito mais informal, pois os BDs NoSQL permitem que sejam armazenados quaisquer dados sobre uma chave ou uma coluna escolhida. Além disso, permitem que sejam adicionados, livremente, campos aos registros do BD, sem ter de definir primeiro quaisquer mudanças na estrutura, como ocorre nos BDs relacionais (SADALAGE; FOWLER, 2012). A ausência de esquema evita o problema de registros com vários elementos sem valor, quando estes não são uniformes (cada registro possui um conjunto diferente de dados), permitindo que cada registro contenha apenas o necessário — nada a mais, nada a menos.

No entanto, a ausência de um esquema explícito, não acarreta a ausência total de esquema, visto que sempre há um esquema implícito nos dados e em aplicações de BDs (RUIZ; MORALES; MOLINA, 2015).

[2] https://www.w3.org/XML/
[3] http://json.org/
[4] http://bsonspec.org/
[5] https://db-engines.com/en/ranking
[6] https://www.mongodb.com/
[7] https://aws.amazon.com/pt/dynamodb/
[8] https://www.couchbase.com/
[9] http://couchdb.apache.org/

1.1 Motivação

A flexibilidade de esquema é uma força motriz por trás da popularidade dos BDs NoSQL (SADALAGE; FOWLER, 2012). Embora essa flexibilidade de esquema pareça ser um ponto positivo, a informação sobre o esquema é essencial, por exemplo, durante o desenvolvimento de uma aplicação. Caso contrário, o acesso aos dados torna-se mais oneroso.

A ausência de informações referentes ao esquema não apenas impede a aplicação de ferramentas de integração, mas também dificulta várias tarefas de processamento de dados, como busca, manipulação, otimização, tradução ou evolução (WISCHENBART et al., 2012).

Na maioria das vezes, as informações de esquema de BDs NoSQL estão implícitas no código da aplicação, o que pode se tornar um grande problema se múltiplos aplicativos, desenvolvidos por pessoas diferentes, acessarem o mesmo BD (SADALAGE; FOWLER, 2012). Sendo assim, é preciso ter alguma noção confiável de sua estrutura, pois sem esse conhecimento, o desenvolvimento de aplicações ou a análise de dados, torna-se um trabalho custoso e, às vezes, impraticável (KLETTKE; STÖRL; SCHERZINGER, 2015).

Desta motivação, surge como propósito do presente Trabalho de Conclusão de Curso (TCC), o desenvolvimento de uma ferramenta que, dada uma coleção de documentos no formato JSON, armazenados em um BD NoSQL orientado a documentos, extraía o esquema implícito na coleção, facilitando assim o desenvolvimento de sistemas, a integração e validação de dados, bem como a análise dos mesmos.

Este TCC é parte integrante de um projeto de pesquisa com o tema “Engenharia reversa de Bancos de Dados NoSQL”, em desenvolvimento no GBD - Grupo de Pesquisa em Banco de Dados da UFSC. Este projeto de pesquisa propõe uma metodologia para extrair e integrar esquemas de BDs NoSQL e a geração de visões globais RDF (*Resource Description Framework*)[10] dos esquemas integrados.

[10] https://www.w3.org/RDF/

1.2 Objetivos

1.2.1 Objetivo Geral

O presente TCC tem por objetivo geral o desenvolvimento de uma ferramenta que possibilite a extração do esquema de uma coleção de documentos JSON persistidos em um BD NoSQL orientado a documentos.

1.2.2 Objetivos Específicos

Os objetivos específicos necessários para se alcançar o objetivo geral são os seguintes:

- Extrair os esquemas brutos (*raw schema*) de documentos JSON armazenados em BD NoSQL orientados a documentos;
- Mapear os esquemas brutos (*raw schema*) para um único esquema no padrão JSON *Schema*;
- Avaliar e validar a ferramenta proposta através de estudos de caso.

1.3 Justificativa

Conhecer a estrutura de uma coleção de dados armazenados em um BD NoSQL ou seja, o esquema, é essencial durante o desenvolvimento de aplicações. Por exemplo, várias aplicações que lidam com dados geográficos, como o *Foursquare*, armazenam esses dados em formato JSON, mas não definem um esquema para os mesmos, dificultando assim consultas a esses dados pelos usuários, pois estes desconhecem a estrutura dos documentos. Ainda existem poucas ferramentas no mercado para esse propósito e, normalmente, as mesmas não adotam um formato padrão para representação do esquema gerado.

Sendo assim, a ferramenta proposta nesse TCC tem como objetivo gerar um esquema único de uma coleção de documentos no formato JSON (formato padrão para intercâmbio de dados complexos e heterogêneos (IETF, 2014)), armazenados em um BD NoSQL orientado a documentos. O esquema gerado segue a definição JSON *Schema* — essa definição está atualmente em seu sexto *draft* (JSON Schema, 2017) e deve se

estabelecer como um padrão para especificação de esquemas JSON — e pode auxiliar no desenvolvimento de aplicações, na integração e na validação de dados, bem como na análise de uma coleção de dados.

1.4 Metodologia

A primeira etapa do trabalho consistiu em um levantamento do estado da arte sobre *"extração de esquemas de BDs NoSQL"*. Para isto, foi realizada a leitura de livros, artigos, *surveys* e publicações pertinentes ao assunto. A relevância das obras encontradas foi definida em conjunto com o orientador e coorientador. A partir desse levantamento, foi feito um estudo teórico sobre as técnicas e tecnologias envolvidas no processo de extração de esquemas de BDs NoSQL, mais especificamente em BDs orientados a documentos.

A segunda etapa focou na análise dos trabalhos relacionados ao tema abordado, visando realizar o levantamento de requisitos e de características fundamentais para um processo de extração de esquemas, com o objetivo de se propor uma nova ferramenta que ofereça um diferencial em relação aos trabalhos relacionados.

A terceira etapa consistiu no levantamento de requisitos, escolha das tecnologias a serem utilizadas na implementação, definição da arquitetura e o desenvolvimento da ferramenta proposta.

Na quarta etapa, foram realizados estudos de caso a fim de avaliar a qualidade do JSON *Schema* extraído. Foi realizada também a análise de tempo de processamento da ferramenta para extração do esquema de grandes *datasets* da vida real e, por fim, a análise comparativa entre a ferramenta proposta e alguns trabalhos relacionados.

1.5 Estrutura do trabalho

O presente trabalho se divide em seis capítulos. Este primeiro apresenta os objetivos, a metodologia e o contexto de surgimento deste TCC. No capítulo 2 é apresentada a revisão dos fundamentos teóricos que servem como base ao entendimento e desenvolvimento da ferramenta proposta. O capítulo 3 reúne uma análise dos trabalhos relacionados à ferramenta proposta. O processo de desenvolvimento da ferramenta é descrito no capítulo 4, quando também são explicadas as decisões tecnológicas, as bibliotecas utilizadas, a

arquitetura escolhida, bem como o processo de extração de esquema. O capitulo 5, por sua vez, apresenta três estudos de caso, bem como a análise dos resultados obtidos. Por fim, a conclusão (capítulo 6) reforça a contribuição deste trabalho e aponta alguns caminhos para a continuidade do projeto em trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta uma revisão dos conceitos essenciais para o entendimento e desenvolvimento do presente trabalho. São explicados conceitos sobre *Big Data* (seção 2.1), Teorema CAP (seção 2.2), NoSQL (seção 2.3), JSON (seção 2.4) e JSON *Schema* (seção 2.5).

2.1 *Big Data*

Big Data é um *buzzword* utilizado para definir um grande volume de dados não estruturados — na faixa de *exabytes* (10^{18}) ou mais — que requer novas tecnologias e arquiteturas para que seja possível seu processamento, gerenciamento e armazenamento de forma efetiva e eficiente (SADALAGE; FOWLER, 2012).

Grandes empresas, como a *Google* e a *Amazon*, lidaram com grandes volumes de dados e com um tráfego cada vez maior no início do século XX. Escalar verticalmente se tornou inviável, pois adquirir máquinas maiores — com maior poder de processamento e com maior capacidade de armazenamento — significava maiores custos, sem mencionar que há limites físicos quanto ao aumento do tamanho (HAN et al., 2011).

A solução para processar esse grande volume de dados foi escalar horizontalmente, ou seja, utilizar mais máquinas, com poder de processamento menor, de forma distribuída. No entanto, essas empresas se depararam com um novo problema: a incompatibilidade entre os BDs relacionais e *clusters*, pois os BDs relacionais não foram projetados para serem executados de forma distribuída, sendo assim, não suportavam facilmente a fragmentação de dados.

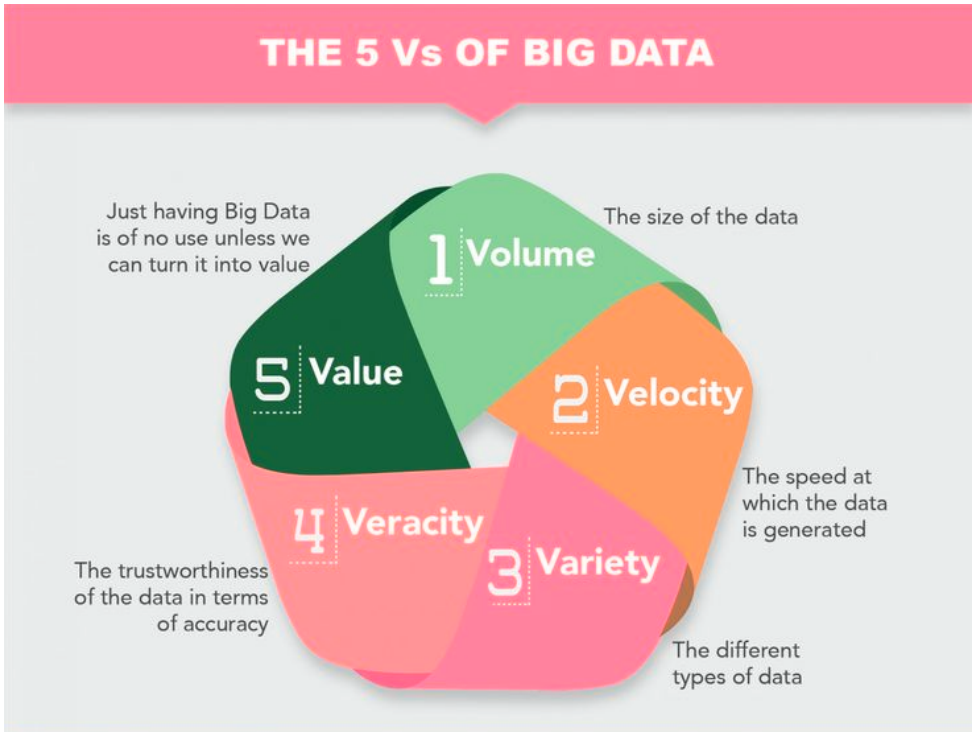
Essa incompatibilidade fez com que essas empresas desenvolvessem soluções próprias para o armazenamento de grandes volumes de dados. *BigTable*[1] e *Dynamo*[2] foram as soluções propostas pela *Google* e *Amazon*, respectivamente, com suporte a fragmentação e replicação de dados (SADALAGE; FOWLER, 2012).

O volume de dados é apenas uma das propriedades de *Big Data*, outras propri-

[1] https://cloud.google.com/bigtable/
[2] https://aws.amazon.com/pt/dynamodb/

edades são: variedade, velocidade, valor e veracidade (Figura 1). Essas propriedades são conhecidas como *"os 5 Vs de Big Data"* (Big Data is the new black, 2016):

Figura 1: Os 5 Vs de *Big Data*



Fonte: Big Data is the new black (2016)

Cada propriedade representada na Figura 1 é brevemente expressa abaixo:

1. *Volume*: Refere-se ao volume de dados, que é força motriz por trás do conceito de *Big Data*, já visto anteriormente;
2. *Velocidade*: Refere-se à velocidade em que os dados são gerados, transmitidos e agregados de diversas fontes;
3. *Variedade*: Trata a variedade de estrutura dos dados — dados estruturados, semi-estruturados e não estruturados (KATAL; WAZID; GOUDAR, 2013) —, pois os mesmos podem ser provenientes de diferentes fontes de dados, como páginas *web*, *logs*, redes sociais, *e-mail*, documentos, dados de sensores, entre outros (SAGIROGLU; SINANC, 2013);
4. *Veracidade*: Está relacionado à autenticidade dos dados, pois com um grande volume de dados sendo gerados em alta velocidade, muitas informações podem ser irrelevantes. Sendo assim, é necessário realizar filtros e deles extrair o que realmente for relevante (Big Data is the new black, 2016);

5. *Valor*: Refere-se ao valor gerado por *Big Data*, pois para garantir que os outros 4 Vs tragam rentabilidade, é essencial que os resultados de *Big Data* agreguem valor para a organização (KATAL; WAZID; GOUDAR, 2013).

Segundo Kaisler et al. (2013, p. 1), um grande desafio para os pesquisadores e para os profissionais de TI é que a taxa de crescimento de *Big Data* ultrapassa rapidamente a capacidade de conceber sistemas adequados para lidarem com os dados de forma eficaz, bem como ultrapassam também a capacidade de analisar esses dados para extrair informações e conhecimentos úteis para a tomada de decisão.

2.1.1 *Map-Reduce*

Quando se fala em desenvolvimento de aplicações para *Big Data*, uma das principais técnicas de programação utilizadas é o *Map-Reduce*, que proporciona a paralelização do processamento. Segundo Sadalage e Fowler (2012, p. 98):

> O padrão *map-reduce* é uma forma de organizar o processamento de maneira a aproveitar as múltiplas máquinas de um *cluster*. Ao mesmo tempo, mantém-se o quanto for possível do processamento e dos dados de que ele precisa na mesma máquina.

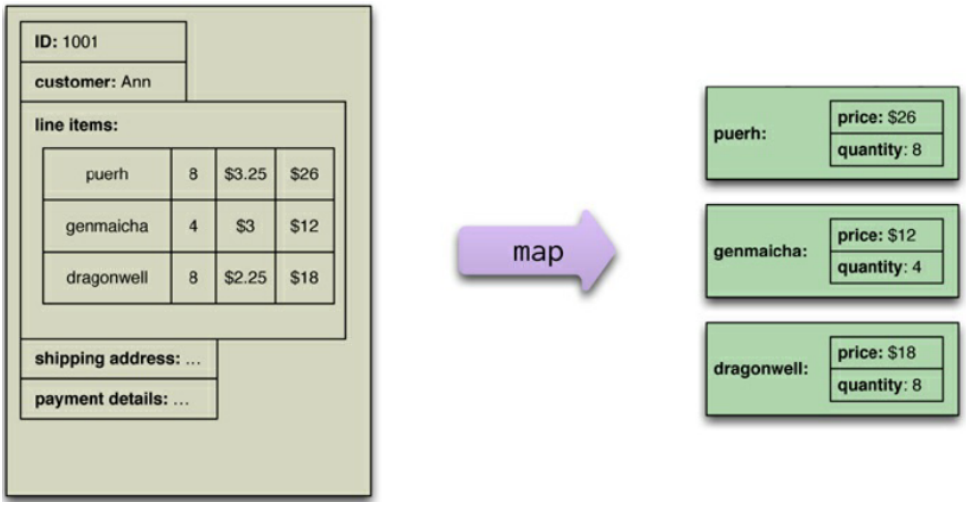
O padrão destacou-se pela primeira vez com o *framework MapReduce* do *Google*. *MapReduce* é um *framework* de programação para computação distribuída, que utiliza o método dividir para conquistar. *MapReduce* divide problemas complexos de *Big data* em pequenas unidades de trabalho, processando-as em paralelo (SAGIROGLU; SINANC, 2013). Segundo Sadalage e Fowler (2012), uma operação de *map-reduce* pode ser dividida em duas etapas:

- *Etapa de mapeamento*: Função cuja entrada é um único documento e cuja saída são alguns pares de chave-valor correspondendo aos itens;
- *Etapa de redução*: Função cuja entrada são múltiplos valores para uma mesma chave e cuja saída são os valores combinados.

A Figura 2 mostra a etapa de mapeamento, na qual a entrada é um documento que representa um pedido e a saída são os produtos do pedido (*puerth*, *genmaicha* e

dragonwell). A chave, do par chave-valor, representa o nome do produto e o valor, por sua vez, contém as informações de preço (*price)* e quantidade (*quantity*) do produto no pedido.

Figura 2: Etapa de mapeamento



Fonte: Sadalage e Fowler (2012, p. 99)

A Figura 3 mostra uma etapa de redução, na qual a entrada são múltiplos valores (*price* e *quantity*) para uma mesma chave (*puerth*) e a saída é a combinação dos valores.

Figura 3: Etapa de redução

Fonte: Sadalage e Fowler (2012, p. 100)

2.2 Teorema CAP

Segundo Brewer (2000), um sistema distribuído tem três propriedades desejáveis:

- *Consistência* (C) – equivalente a ter uma única cópia atualizada dos dados;

- *Alta disponibilidade* (A) desses dados (para atualizações);
- *Tolerância* às Partições de rede (P) – o sistema continua funcionando, mesmo quando um problema ocorre na rede, dividindo o sistema em duas ou mais partições.

No entanto, Brewer (2000) afirma que um sistema distribuído só pode garantir duas entre essas três propriedades simultaneamente.

A Figura 4 mostra as relações entre consistência, disponibilidade e tolerância à partição de rede. Por exemplo, um sistema baseado em CA, garante consistência e alta disponibilidade. Porém, não consegue ser tolerante à partições de rede. Essa característica é comum entre os BDs do modelo relacional. Um sistema baseado em PA garante alta disponibilidade e tolerância à partições de rede. No entanto, não consegue garantir consistência, pois um dado em um nodo qualquer pode não ser o dado mais atualizado. Um sistema baseado em CP, garante consistência e tolerância à partições de rede. Porém, não consegue garantir alta disponibilidade, pois para manter a consistência, na atualização de um dado, é preciso bloquear o acesso ao mesmo até que a atualização do dado seja refletida (propagada) em todos os nós do sistema.

Figura 4: Teorema CAP

Fonte: Brewer (2000, p. 4)

Como em sistemas distribuídos, ser tolerante às partições de rede é essencial, decidir entre consistência e disponibilidade torna-se uma escolha não necessariamente binária. Sendo assim, deve-se otimizar a consistência e a disponibilidade para, então, conseguir algum *trade-off*[3] das três propriedades (BREWER, 2012).

[3] *Trade-off*: Perder uma qualidade ou aspecto de algo, ganhando em troca outra qualidade ou aspecto.

2.3 NoSQL

Os BDs NoSQL surgiram como uma solução aos problemas que grandes empresas enfrentaram no começo do século XX, quanto ao processamento e armazenamento de grandes volumes de dados.

Segundo Tudorica e Bucur (2011, p. 1), um BD NoSQL pode ser definido como:

> "Um sistema de banco de dados que é distribuído, pode não exigir esquemas de tabela fixa, geralmente evitam operações de *join*, normalmente escalam horizontalmente, não expõe uma interface SQL e podem ser *open source*".

Ao oferecer alto desempenho e alta disponibilidade, esses BDs não priorizam a consistência, pois, de acordo com o teorema CAP (ver seção 2.2), alta disponibilidade e consistência, são propriedades que não são possíveis de serem garantidas de forma absoluta e simultânea, quando se opta por tolerância a partição de rede, que é o caso de sistemas distribuídos (BREWER, 2000).

Por não priorizar a consistência, esses bancos de dados acabam perdendo as características ACID dos BDs relacionais, em troca de manter as propriedades BASE. Segundo Brewer (2012), um sistema BASE se concentra em alta disponibilidade, enquanto que um sistema ACID se concentra em consistência. Resumidamente, um sistema BASE é um sistema que funciona basicamente todo o tempo (***B**asically **A**vailable*) e não precisa ser consistente todo o tempo (***S**oft-state*), pois em um momento indeterminado, o sistema estará em um estado consistente (***E**ventually Consistent*). Assim, esses BDs garantem que as atualizações são eventualmente propagadas para todos os nós, mas que existem garantias limitadas sobre a consistência das leituras. Com isso, esses BDs conseguem garantir alto desempenho em operações de leitura e escrita, além de garantir alta disponibilidade e escalabilidade (SADALAGE; FOWLER, 2012).

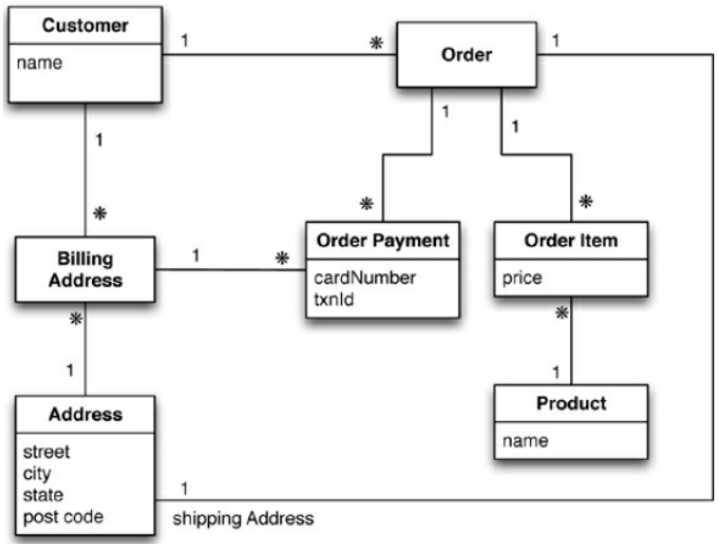
2.3.1 Modelos de dados

O modelo relacional predominou por muitos anos como um modelo de dados padrão para BDs. Esse modelo pode ser descrito por um conjunto de tabelas (Figura 6), constituída por linhas e colunas. As linhas, também chamadas de *tuplas*, representam entidades de interesse e as colunas, por sua vez, representam as características (propriedades)

das entidades. Uma coluna contém um único valor e este pode se referir a outra linha na mesma tabela ou em uma tabela diferente, caracterizando assim um relacionamento entre entidades (SADALAGE; FOWLER, 2012).

A Figura 5 apresenta um exemplo de modelo de dados relacional, utilizado por um *website* de comércio eletrônico hipotético.

Figura 5: Modelo de dados relacional



Fonte: Sadalage e Fowler (2012, p. 37)

A Figura 6 mostra alguns dados de exemplo para o modelo apresentado na Figura 5, onde os mesmos estão devidamente normalizados, de modo que nenhum dado se repete em tabelas múltiplas.

No contexto de BDs NoSQL, não há um único modelo de dados padrão utilizado. Cada solução adota um modelo de dados diferente e os mesmos podem ser classificados em quatro categorias amplamente utilizadas: chave-valor, orientado a documento, orientado a colunas e orientado a grafos (Figura 7) (HASHEM; RANC, 2016).

2.3.1.1 Modelo de dados orientado a agregados

No modelo relacional, as informações a serem armazenadas são divididas em um conjunto de *tuplas*. Uma *tupla* é uma estrutura de dados que contém um conjunto de pares chave-valor, não permitindo o aninhamento de outras *tuplas* (uma *tupla* dentro de outra) e nem de lista de valores.

Figura 6: Dados em um modelo de dados relacional

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

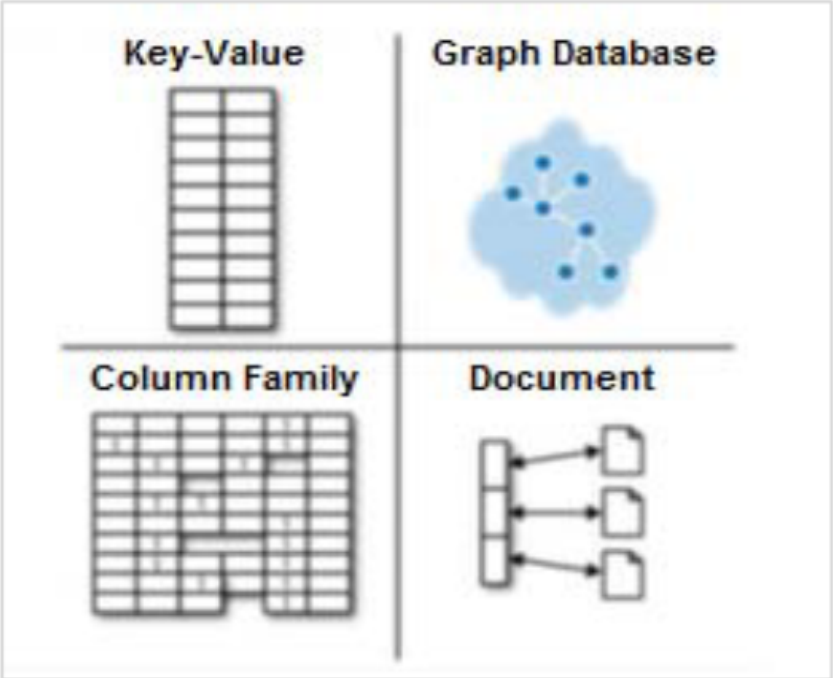
Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Fonte: Sadalage e Fowler (2012, p. 38)

Figura 7: Modelos de dados de banco de dados NoSQL

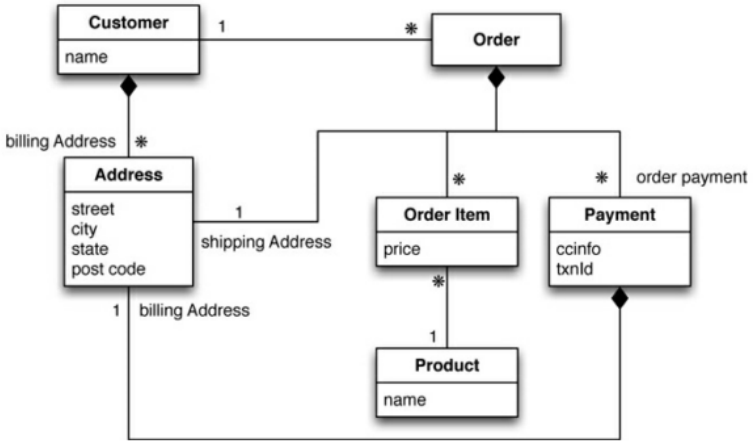


Fonte: Hashem e Ranc (2016, p. 1)

Na orientação agregada — utilizada pelos modelos de dados chave-valor, orientado a documento e orientado a colunas — os dados são tratados como unidades e tem uma estrutura mais complexa do que uma *tupla*. Um agregado permite o aninhamento de listas e de outras estruturas de dados (SADALAGE; FOWLER, 2012). A Figura 8a mostra o respectivo modelo de dados agregado para o exemplo demonstrado na Figura 5 e

a Figura 8b mostra o exemplo da Figura 6 em um modelo de dados agregado.

Figura 8: Modelo de dados agregado e dados agregados representado em JSON



(a) Modelos de dados agregado

```
// in customers
{
  "billingAddress": [
    {
      "city": "Chicago"
    }
  ],
  "id": 1,
  "name": "Martin"
}

// in orders
{
  "customerId": 1,
  "id": 99,
  "orderItems": [
    {
      "price": 32.45,
      "productId": 27,
      "productName": "NoSQL Distilled"
    }
  ],
  "orderPayment": [
    {
      "billingAddress": {
        "city": "Chicago"
      },
      "ccinfo": "1000-1000-1000-1000",
      "txnId": "abelif879rft"
    }
  ],
  "shippingAddress": [
    {
      "city": "Chicago"
    }
  ]
}
```

(b) Dados em um modelo de dados agregado

Fonte: Sadalage e Fowler (2012, p. 39) (Adaptado)

2.3.2 Bancos de dados chave-valor

Esses BDs são os menos complexos, pois todos os dados consistem, basicamente, em uma chave indexada e um valor. Conhecidos também como tabelas de *hash* distribuídas, permitem uma velocidade de consulta maior do que uma consulta em um BD relacional (JMTAURO; S; A.B, 2012).

O valor, do par chave-valor, é um BLOB[4] (*Binary Large OBject*) que o BD apenas armazena, sem se preocupar em saber o que há dentro dele (SADALAGE; FOWLER, 2012).

Além de suportar armazenamento em massa, os BDs chave-valor oferecem um alto desempenho em operações de leitura e escrita concorrentes. Alguns dos BDs

[4] Coleção de dados binários armazenados como uma única entidade.

chave-valor mais populares, segundo *DB-Engines Ranking* são: *Redis*[5], *MemcachedDB*[6] e *Hazelcast*[7].

2.3.3 Bancos de dados orientados a colunas

Os BDs orientados a colunas, conhecidos também como BDs colunares ou BDs famílias de colunas, oferecem um alto desempenho e uma arquitetura altamente escalável. Esses BDs podem ser considerados como um tipo específico de BD chave-valor. Porém, são mais complexos e, neste caso, muda-se o paradigma de orientação a registros ou *tuplas*, como utilizado no modelo relacional, para orientação a atributos ou colunas.

BDs colunares utilizam, como modelo de dados, uma tabela com um conjunto predefinido de colunas, não suportando associação entre tabelas, como ocorre no modelo relacional (HAN et al., 2011). As tabelas são constituídas por colunas, famílias de colunas e supercolunas.

A coluna é uma unidade atômica de informação, sendo expressa como um par chave-valor. Supercoluna é o agrupamento de colunas que podem ser obtidas juntas ou que possuem uma associação semântica entre si. Família de colunas é o agrupamento de colunas e de supercolunas à uma única chave, sendo a semelhança mais próxima da tabela no modelo relacional (INDRAWAN-SANTIAGO, 2012).

A estrutura de um BD colunar equivale ao esquema do BD. No entanto, esse esquema é mais flexível — se comparado ao esquema de um BD relacional — quanto à adição de uma nova coluna ou de uma nova supercoluna. BDs populares nesta categoria, segundo *DB-Engines Ranking* são: *Cassandra*[8]. *HBase*[9] e *Microsoft Azure Cosmos DB*[10].

2.3.4 Bancos de dados orientados a grafos

Um BD orientado a grafos é um BD que utiliza grafos como o meio para representar seu esquema. Seus componentes básicos são: os nós (vértices), os relacionamentos (arestas) e as propriedades (atributos) dos nós e dos relacionamentos (JMTAURO; S; A.B,

5 https://redis.io/
6 http://memcachedb.org/
7 https://hazelcast.com/
8 https://cassandra.apache.org/
9 http://hbase.apache.org/
10 https://azure.microsoft.com/pt-br/services/cosmos-db/

2012). Neste caso, o BD pode ser visto como um multigrafo rotulado e direcionado, em que cada par de nós pode ser conectado por mais de uma aresta.

Os BDs orientados a grafos são projetados para dados altamente complexos e conectados, que ultrapassam as capacidades de relacionamentos com *joins* dos BDs relacionais. São frequentemente bons para encontrar pontos comuns e anomalias em grandes conjuntos de dados. Permitem descobrir padrões que são difíceis de detectar ao usar representações tradicionais, como tabelas. Os principais BDs orientados a grafos, segundo *DB-Engines Ranking* são: *Neo4J*[11], *Microsoft Azure Cosmos DB*[12] e *OrientDB*[13].

2.3.5 Bancos de dados orientados a documentos

Os BDs orientados a documentos, ou simplesmente BDs de documentos, não estão preocupados com o alto desempenho em operações de leitura e escrita concorrentes, mas sim, em garantir um armazenamento eficiente de grandes volumes de dados e um bom desempenho em operações de consulta (HAN et al., 2011). Sendo assim, BDs de documentos tentam melhorar a disponibilidade replicando os dados, utilizando uma configuração do tipo mestre-escravo. (SADALAGE; FOWLER, 2012). Os mesmos dados ficam disponíveis em múltiplos nodos, os quais os clientes podem acessar mesmo quando o nodo primário estiver indisponível.

Esses BDs podem ser considerados como uma subcategoria dos BDs chave-valor. Sendo a diferença entre esses dois, o fato de que, enquanto os BDs chave-valor armazenam valores como BLOB, os BDs de documentos armazenam os valores como documentos, os quais podem ser armazenados em um formato de troca de dados padrão, como XML[14], JSON[15] (ver seção 2.4) ou BSON[16] (ver seção 2.4.1) (HASHEM; RANC, 2016; INDRAWAN-SANTIAGO, 2012; HAN et al., 2011).

Um dos recursos interessantes dos BDs de documentos, comparados aos BDs chave-valor, é a possibilidade de consultar os dados dentro dos documentos sem ter que recuperar o documento inteiro por sua chave e depois examiná-lo (SADALAGE; FOWLER, 2012).

11 https://neo4j.com/
12 https://azure.microsoft.com/pt-br/services/cosmos-db/
13 http://orientdb.com/
14 https://www.w3.org/XML/
15 http://json.org/
16 http://bsonspec.org/

Em geral, BDs de documentos não possuem esquema, por isso, o número de campos não é limitado e novos campos podem ser adicionados dinamicamente a um documento. Um documento, por sua vez, pode ser representado como uma estrutura de dados em forma de árvore, constituída por valores escalares (*string*, *boolean*, numérico), por documentos aninhados e coleções. Como outros BDs NoSQL, BDs de documentos também não fornecem propriedades de transação ACID. BDs populares nessa categoria, segundo *DB-Engines Ranking* são: *MongoDB*[17], *CouchDB*[18] e *Couchbase*[19].

2.3.5.1 *MongoDB*

MongoDB é um SGBD (Sistema de Gerenciamento de Banco de Dados) NoSQL orientado a documentos, lançado em 2009. Cada instância do *MongoDB* possui múltiplos BDs e cada BD pode ter múltiplas coleções, as quais não possuem esquemas de tabelas predefinidas. Os dados são armazenados como documentos BSON (objetos JSON codificados em binário) (GYORODI et al., 2015) (ver seção 2.4.1) e suas principais características são: flexibilidade, poder, velocidade e facilidade de uso. O *MongoDB* suporta servidores replicados e indexação, além de oferecer *drivers* para várias linguagens de programação (BOICEA; RADULESCU; AGAPIN, 2012).

A ferramenta proposta utiliza o *MongoDB* tanto como fonte de origem dos documentos para o processo de extração, quanto para armazenamento dos esquemas gerados. O *MongoDB* foi escolhido como SGBD da ferramenta, pois atualmente está classificado entre os cinco principais SGBDs e na primeira posição entre os SGBDs NoSQL[20].

2.4 JSON

JSON é um formato leve utilizado para intercâmbio de dados complexos e heterogêneos. Por ser um formato de texto completamente independente de linguagem (ECMA-404, 2014), utilizando convenções que são familiares àquelas utilizadas em linguagens como C, C++, C#, *Java*, *JavaScript*, *Perl* e *Python*, o formato JSON foi amplamente adotado como um formato ideal para intercâmbio de dados.

[17] https://www.mongodb.com/
[18] http://couchdb.apache.org/
[19] https://www.couchbase.com/
[20] Disponível em: <https://db-engines.com/en/>. Acesso em: 25 set. 2017.

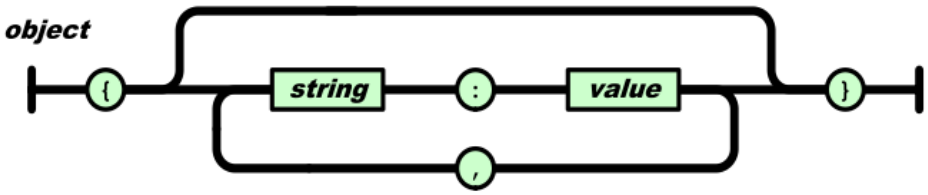
JSON é construído basicamente em duas estruturas, presentes em muitas linguagens de programação:

- Uma coleção de pares chave-valor, semelhante a um objeto.
- Uma lista ordenada de valores, semelhante a um *array*.

A terminologia que define a estrutura de um documento JSON compreende os seguintes conceitos:

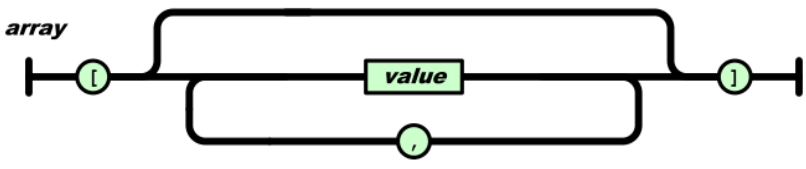
- *Objeto*: conjunto desordenado de pares chave-valor. Um objeto, representado na Figura 9, começa com '{' (abre-chaves) e termina com '}' (fecha-chaves), cada chave é seguida de ':' (dois-pontos) e os pares chave-valor são separados por ',' (vírgula);
- *Array*: coleção ordenada de valores. Um *array*, representado na Figura 10, começa com '[' (abre-colchetes) e termina com ']' (fecha-colchetes) e os valores são separados por ',' (vírgula);
- *Valor*: um valor JSON pode ser um valor de tipo primitivo (*string*, numérico, booleano (*true*, *false*)), de tipo estruturado (objeto, *array*) ou nulo *null*), como representado na Figura 11. O uso de objetos e *arrays* como valores permite definir estruturas aninhadas.

Figura 9: Representação de um objeto em JSON



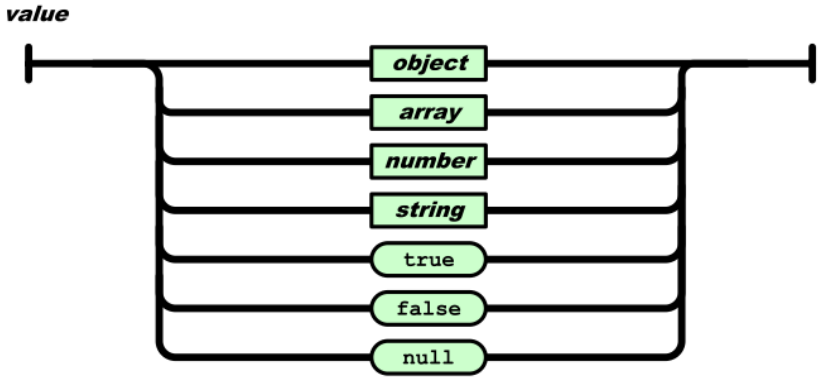
Fonte: ECMA-404 (2014)

Figura 10: Representação de um *array* em JSON



Fonte: ECMA-404 (2014)

Figura 11: Representação de um valor em JSON



Fonte: ECMA-404 (2014)

Por ser facilmente compreendido por desenvolvedores e máquinas, o JSON tornou-se o formato mais popular para enviar requisições à APIs (*Application Programming Interface*) e respostas sobre o protocolo HTTP (*Hypertext Transfer Protocol*), desempenhando assim um papel fundamental no contexto das aplicações *web* (PEZOA et al., 2016).

2.4.1 BSON

BSON[21], abreviação de *Binary* JSON, é uma serialização codificada em binário de documentos JSON. Como JSON, BSON suporta a incorporação de documentos e *arrays* em outros documentos e *arrays*. BSON também é considerado uma extensão do JSON que permitem a representação de tipos de dados que não fazem parte da especificação JSON. Por exemplo, BSON possui os tipos *Date*, *Timestamp*, *Binary*, *ObjectID*, *RegExp*, *Long*, *Undefined*, *DBRef*, *Code*, *MinKey* e *MaxKey*. A Figura 12 representa um documento JSON em BSON.

Figura 12: Representação de um documento JSON em BSON

```
                                \x16\x00\x00\x00                // total document size
                                \x02                            // 0x02 = type String
{"hello": "world"}          →   hello\x00                       // field name
                                \x06\x00\x00\x00world\x00       // field value
                                \x00                            // 0x00 = type EOO ('end of object')
```

Fonte: bson-spec-org (2017)

[21] http://bsonspec.org/

2.5 JSON *Schema*

Com a popularidade do formato JSON (ver seção 2.4) nas aplicações *web*, nota-se que em muitos cenários, uma forma declarativa de se especificar um esquema para documentos JSON pode ser algo benéfico. Com a definição de um esquema, pode-se evitar que uma API receba chamadas mal formadas, que podem ocasionar erros internos, caso alguma informação do documento não seja validada pela API (PEZOA et al., 2016).

A definição de um esquema também evita que a validação de um documento JSON seja feita durante a codificação da API, pois o esquema definido vai servir como um filtro que aceita somente documentos que estão no formato correto. Essa definição de esquema também serve como um meio de se padronizar a especificação de quais tipos de documento JSON são aceitos como entrada e saída de uma API.

A Figura 13 representa o corpo de uma requisição HTTP (*body*), no formato JSON, a uma API de um *web service*, que provê informações hipotéticas sobre restaurantes ao redor do mundo. A documento JSON, contido no corpo da requisição, possui as propriedades *latitude* e *longitude*, das quais se deseja obter informações.

Figura 13: Corpo de uma requisição HTTP a uma API de um *web service* hipotético

```
{
  "latitude": -73.9574128,
  "longitude": 40.7701235
}
```

Fonte: Autor (2017)

A Figura 14 representa a resposta da API, no formato JSON, para a requisição. O documento JSON, representa um objeto JSON (ver seção 2.4), contendo os atributos *borough*, *cuisine* e *name*, ambos do tipo *string*. O documento também possui um atributo *address*, do tipo objeto que, por sua vez, possui os atributos de tipo *string*: *street* e *zipcode*, além do atributo *coord*, sendo do tipo *array*. O documento possui também uma propriedade *_id*, que representa o id do documento no BD utilizado pelo *web service*.

O JSON *Schema*[22] é uma tentativa de fornecer uma linguagem de esquema de propósito geral para JSON, que permita aos usuários restringir a estrutura dos documentos JSON — nomes necessários em objetos, os tipos permitidos para um atributo de um

[22] http://json-schema.org/

Figura 14: Resposta da API a uma requisição


```
{
  "_id": {
    "$oid": "59978755a065669705073544"
  },
  "address": {
    "coord": [
      -73.9574128,
      40.7701235
    ],
    "street": "East 74 Street",
    "zipcode": "10021"
  },
  "borough": "Manhattan",
  "cuisine": "Italian",
  "name": "Cucina Vivolo"
}
```


Fonte: Autor (2017)

objeto ou para valores em um *array*, além de oferecer uma estrutura para verificar a integridade das solicitações e sua conformidade com a API (PEZOA et al., 2016). Além disso, o JSON *Schema* permite que sejam incorporadas descrições nas especificações do esquema, permitindo que o autor do esquema forneça comentários em linha.

A Figura 15 representa o JSON *Schema* para validar o corpo das requisições a API. Todas as requisições a API devem informar um documento JSON, com os atributos *latitute* e *longitude*, ambos do tipo *number* (*"type":"number"*), sendo que não é permitido atributos adicionais na requisição (*"additionalProperties":false*), além de ambos atributos serem obrigatórios (*"required":["latitude","longitude"]*).

Figura 15: JSON *Schema* para validar as requisições a API de um *web service* hipotético

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "properties": {
    "latitude": {
      "type": "number"
    },
    "longitude": {
      "type": "number"
    }
  },
  "additionalProperties": false,
  "required": [
    "latitude",
    "longitude"
  ]
}
```

Fonte: Autor (2017)

O documento JSON sendo validado ou descrito (Figura 13), é chamado de instância e o documento que contém a descrição (Figura 15) é chamado de esquema. As propriedades de uma instância podem ser validadas, utilizando a palavra-chave *"properties"*. Cada item em *"properties"*, representa uma propriedade na instância.

Restrições podem ser aplicadas a uma propriedade, adicionando palavras-chave de validação ao esquema. Por exemplo, a palavra-chave *"type"* pode ser usada para restringir uma propriedade a um tipo JSON (ver seção 2.4): *object*, *array*, *string*, *number*, *boolean* ou *null*. Para representar mais de um tipo em uma propriedade, pode-se utilizar a palavra-chave *"anyOf"*, que restringe a propriedade a mais de um tipo, por exemplo: *"anyOf":["string","number"]*.

Uma propriedade pode ser definida como obrigatória, para isso o nome da propriedade deve ser adicionada a um *array*, e atribuído à palavra-chave *"required"*. Uma instância é válida contra essa palavra-chave, se cada item em *"required"* for o nome de uma propriedade na instância.

Propriedades adicionais podem ser permitidas também, utilizando para isso, a palavra-chave *"additionalProperties"*. Quando atribuído o valor *true*, o esquema permite propriedades adicionais, caso contrário, *false*, a instância deve possuir apenas as propriedades definidas em *"properties"*.

Uma propriedade já definida pode ser reaproveitada durante a especificação do esquema, definindo a propriedade em uma seção de definições e referenciando-a durante a especificação do esquema. Para isto, basta declarar a palavra-chave *"definitions"* e atribuir, como valor, um objeto composto por propriedades que podem ser referenciadas durante a especificação do esquema. Para referenciar uma propriedade definida, basta utilizar a palavra-chave *"$ref"*, enquanto seu valor deve apontar para a definição, por exemplo: *"#/definitions/NomeDaPropriedade"*.

A Figura 16 representa a especificação de um JSON *Schema* para a resposta da API do *web service* de restaurantes. O esquema utiliza algumas das palavras-chaves citadas, como, por exemplo: *"definitions"*, *"type"*, *"required"* e *"additionalProperties"*. A palavra-chave *"items"*, utilizada na especificação de propriedades do tipo *array*, restringe os itens do *array* a um tipo especifico, como *"type":"number"*, ou a mais de um tipo, como *"anyOf":["string","number"]*.

Figura 16: JSON *Schema* representando a resposta da API a uma requisição.

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "ObjectID": {
      "type": "object",
      "properties": {
        "$oid": {"type": "string"}
      },
      "required": ["$oid"]
    }
  },
  "properties": {
    "_id": {"$ref": "#/definitions/ObjectID"},
    "address": {
      "type": "object",
      "properties": {
        "building": {"type": "string"},
        "coord": {
          "type": "array",
          "items": {"type": "number"}
        },
        "street": {"type": "string"},
        "zipcode": {"type": "string"}
      },
      "additionalProperties": false,
      "required": ["building","coord","street","zipcode"]
    },
    "borough": {"type": "string"},
    "cuisine": {"type": "string"},
    "name": {"type": "string"}
  },
  "additionalProperties": false,
  "required": ["_id","address","borough","cuisine","name"]
}
```

Fonte: Autor (2017)

3 Trabalhos Relacionados

Este capítulo apresenta e compara as propostas de Klettke, Störl e Scherzinger (2015) (seção 3.1), Ruiz, Morales e Molina (2015) (seção 3.2), Wang et al. (2015) (seção 3.3), Izquierdo e Cabot (2013) (seção 3.4) e Wischenbart et al. (2012) (seção 3.5) para extração de esquemas de dados JSON, armazenados em bancos de dados NoSQL. Estas propostas serviram de base para este trabalho e, ao final, são comparadas com o presente trabalho.

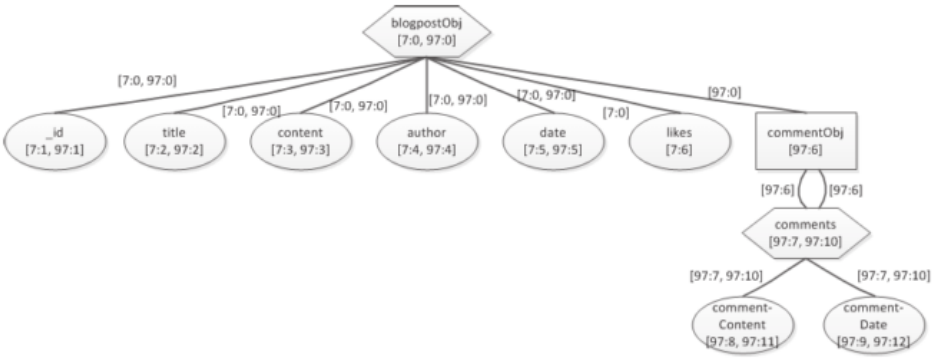
3.1 *Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores*

Os autores propõem um algoritmo para extração de esquemas de BDs NoSQL orientados a documentos *MongoDB* (KLETTKE; STÖRL; SCHERZINGER, 2015). Baseiam-se em ideias propostas para extrair DTDs (*Document Type Definition*) de coleções de documentos XML, mas levam em conta as particularidades dos dados JSON (ver seção 2.4), como a natureza desordenada de seus elementos, além de medidas de similaridade que descrevem a heterogeneidade estrutural dos mesmos.

O algoritmo proposto utiliza uma estrutura de dados em grafo direcionado, representada na Figura 17, que resume a informação estrutural de todos os documentos analisados. Essa estrutura é composta por nós e arestas e permite detectar *outliers* estruturais — padrões que ocorrem em poucos conjuntos de dados —, além de determinar o grau de cobertura dos documentos, capturando a homogeneidade estrutural de uma coleção de documentos. Os nós representam as propriedades de um objeto JSON e as arestas capturam a estrutura hierárquica dos documentos. Ambos são marcados com informações de linhagem, isto é, listas que especificam em que documentos JSON a propriedade estrutural está presente.

Para declarar um esquema e validar dados segundo um esquema, é necessário definir uma linguagem de descrição de esquema ou utilizar alguma que já foi definida. No caso deste artigo, a linguagem de definição de esquema utilizada foi o JSON *Schema* (ver seção 2.5).

Figura 17: Grafo de Identificação de Estrutura proposto por Klettke, Störl e Scherzinger (2015)



Fonte: Klettke, Störl e Scherzinger (2015, p. 433)

O algoritmo é composto por quatro etapas: (*i*) Seleção de documentos; (*ii*) Extração da estrutura dos documentos JSON; (*iii*) Construção do Grafo de Identificação de Estrutura (SG); e, (*iv*) Geração do JSON *Schema*, conforme representado na Figura 18.

Na primeira etapa, a seleção dos documentos pode considerar a coleção completa ou grupos de documentos JSON (distinguidos por propriedades como data, *timestamp*, número de versão). No último caso, pode ser interessante para revelar a evolução do esquema ao longo do tempo, segundo os autores.

Na segunda etapa é extraída a estrutura dos documentos JSONs e a estrutura do SG é definida. O SG é composto por um conjunto finito de nodos e de arestas ou arcos. Os nodos possuem informações como o nome da propriedade que o nodo representa, os documentos em que o nodo ocorre, bem como, o caminho do nodo raiz até o respectivo nodo. Além dessas informações, um nodo contém uma ou mais arestas que direcionam para os nodos filhos ou nodos folhas.

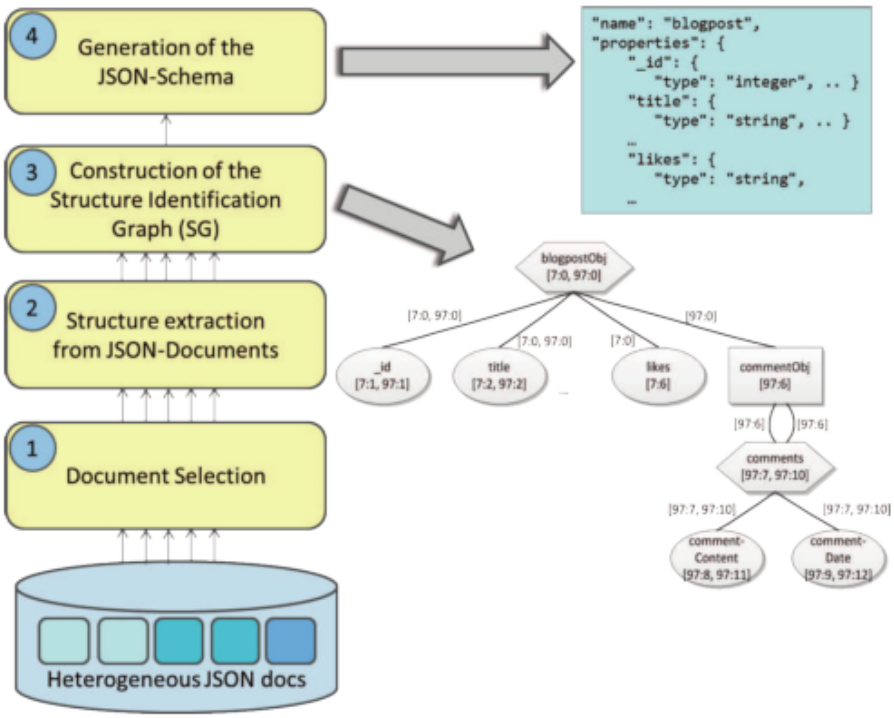
Na terceira etapa é realizada a construção do SG. Primeiro é construído o SG a partir dos dados de entrada do BD. Os nodos no documento JSON são percorridos em *preorder*. Para cada nodo dos documentos de entrada, é realizada a adição ou extensão de um nodo no SG e uma aresta ligando ao nodo pai é adicionada ou estendida.

A adição de um nodo no SG é distinguida em dois casos: se o nodo ainda não existe, então o nodo é adicionado, caso contrário, as informações do nodo atual são anexadas à lista do nodo que já está no SG. A adição de uma aresta é semelhante. Se a

aresta ainda não existe no SG, então ela é adicionada, caso contrário, a lista de arestas do nodo pai é atualizada.

Na quarta etapa, o JSON *Schema* é derivado do SG. Nesta etapa é verificado quais propriedades são obrigatórias e quais são opcionais, bem como, o tipo de dados das propriedades. Para uma propriedade ser considerada obrigatória, ela deve aparecer em todos os documentos em que o nodo pai ocorre. Caso contrário ela é considerada opcional. Uma propriedade pode ter mais de um tipo de dado diferente, neste caso um tipo de dados de união é utilizado, como por exemplo: *"oneOf":[{"type":"string"},{"type" "integer"}].*

Figura 18: Etapas da Extração de esquemas proposta por Klettke, Störl e Scherzinger (2015)



Fonte: Klettke, Störl e Scherzinger (2015, p. 430)

Algumas ideias do trabalho proposto por Klettke, Störl e Scherzinger (2015) contribuíram para o presente trabalho, como, por exemplo:

- Origem dos dados: *Dataset* de um banco NoSQL orientado a documentos;
- Modelo de saída: JSON *Schema*;
- Identificação de propriedades obrigatórias e opcionais;

- Tipo de dados de união, quando uma propriedade possui mais de um tipo de dado diferente.

Quanto às etapas do processo, no presente trabalho não é feita uma seleção de documentos por algum atributo, em prol de obter um esquema final que represente a estrutura de todos os documentos de uma coleção.

3.2 *Inferring Versioned Schemas from NoSQL Databases and its Applications*

Os autores propõem um processo de engenharia reversa para extração de esquemas versionados de BDs NoSQL orientados a agregados (ver seção 2.3.1.1) através da metodologia MDE (*Model Driven Engineering*) (RUIZ; MORALES; MOLINA, 2015).

No contexto desse trabalho, o BD é considerado como um *array* arbitrariamente grande de objetos JSON (ver seção 2.4). Um objeto JSON, por sua vez, deve seguir um formato que seja independente de SGBD NoSQL. Sendo assim, o objeto deve possuir, no mínimo, dois campos padrões:

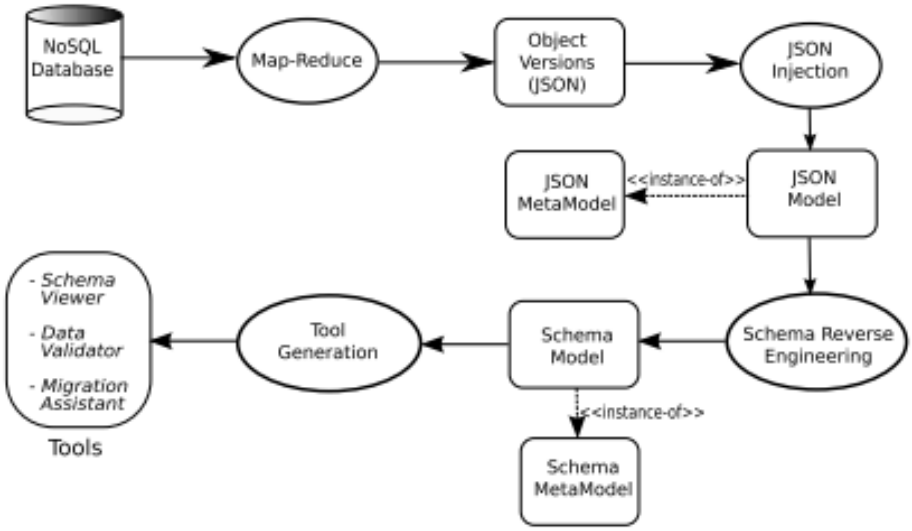
- *type*: campo que descreve o tipo da entidade. Ex.: Contato, Usuário.
- *_id*: campo que representa um identificador exclusivo para o objeto.

O processo de extração de esquemas é composto por três etapas: (*i*) Extração de objetos JSON; (*ii*) Definição do esquema JSON; e, (*iii*) Definição do esquema NoSQL, conforme representado na Figura 19.

Na primeira etapa é realizada uma operação de *map-reduce* (ver seção 2.1.1) para extrair uma coleção de objetos JSON, contendo um objeto para cada versão (estrutura) de uma entidade. Para cada objeto, a operação *map* executa um processo de duas etapas:

1. Gerar o identificador de versão: uma *string* é obtida a partir da concatenação do valor do campo especial *type* com uma representação textual do esquema bruto (*raw schema*) do objeto. A representação do esquema bruto do objeto, representado na Figura 20, por sua vez, é um objeto JSON construído seguindo duas regras:

Figura 19: Etapas da Extração de esquemas proposta por Ruiz, Morales e Molina (2015)



Fonte: Ruiz, Morales e Molina (2015, p. 6)

Figura 20: Exemplo de esquema bruto (*raw schema*)

JSON object	**Raw Schema**
{name:"Omega", city:"Barcelona"}	*{name:String, city:String}*
{title:"Writing and...", *publisher_id:"928672",* *author:{name:"Bradley Holt",* *company:{country:"USA",* *name:"IBM Cloudant"} } }*	*{title:String,* *publisher_id:String,* *author:{name:String,* *company:{country:String,* *name:String} } }*

Fonte: Ruiz, Morales e Molina (2015, p. 7)

a) ter a mesma estrutura que o objeto descrito com relação a campos, objetos aninhados e *arrays*;
b) cada valor primitivo no objeto descrito é substituído no esquema bruto por seu tipo JSON (por exemplo, *string* ou *number*).

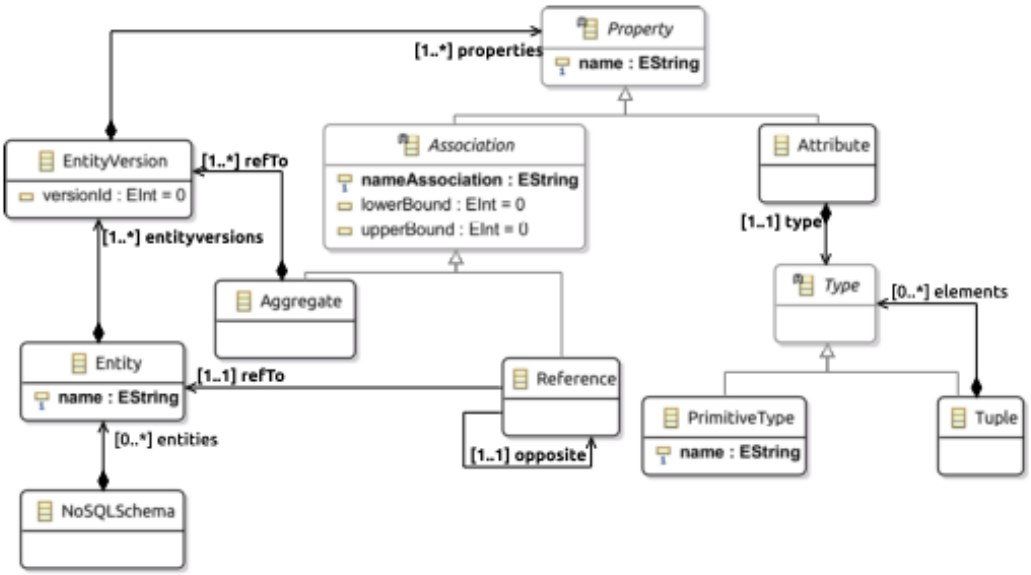
2. Gerar o par de chave-valor *<identificador de versão, objeto>*.

Em seguida, a operação *reduce* seleciona um objeto para cada identificador de versão e adiciona na coleção de objetos JSON que é retornada.

Na segunda etapa, a coleção de objetos JSON obtida na primeira etapa é injetada em um modelo JSON que está em conformidade com um metamodelo JSON.

Na terceira etapa, o processo de engenharia reversa é implementado como

Figura 21: Metamodelo de esquema NoSQL



Fonte: Ruiz, Morales e Molina (2015, p.6)

uma transformação de modelo para modelo (*model-to-model*), cuja entrada é o modelo JSON gerado na segunda etapa e a saída é um modelo que está em conformidade com o metamodelo de esquema NoSQL, representado na Figura 21. O processo de transformação de modelos descobre os elementos do esquema: entidades e versões de entidades, atributos, relacionamentos por agregação e por referência.

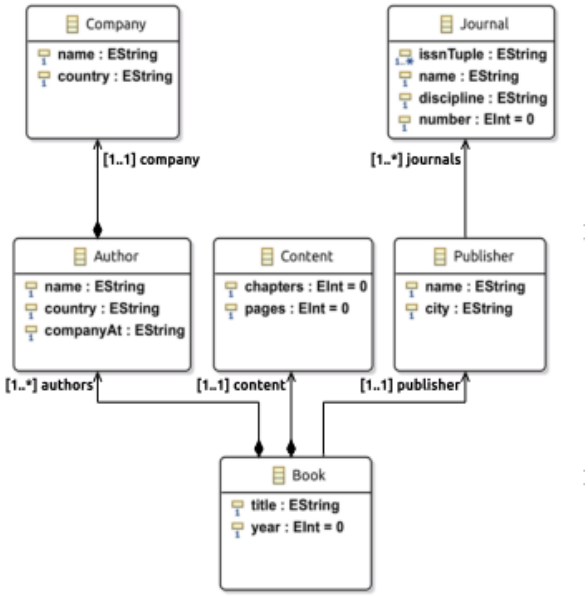
A partir do metamodelo de esquema NoSQL (Figura 21), transformações podem ser aplicadas para gerar o esquema NoSQL em qualquer formato, por exemplo, como um diagrama de classes UML (*Unified Modeling Language*), conforme Figura 22.

Os modelos de esquema NoSQL inferidos podem ser usados para construir utilitários de bancos de dados que exigem conhecimento da estrutura do BD (como SQL *query engines*) e ferramentas auxiliares (como validadores de dados, *scripts* de migração ou diagramas de esquemas).

Algumas ideias do trabalho proposto por Ruiz, Morales e Molina (2015) contribuíram para o presente trabalho:

- Gerar o esquema bruto de um documento JSON (*raw schema*);
- Utilizar uma operação de *map-reduce* para obter um objeto JSON para cada versão (estrutura).

Figura 22: Esquema NoSQL representado como Diagrama UML



Fonte: Ruiz, Morales e Molina (2015, p. 10)

Ao contrário do trabalho proposto por Ruiz, Morales e Molina (2015), o presente trabalho gera um esquema único para uma coleção de documentos JSON, não levando em consideração relacionamentos por referência. Por outro lado, o presente trabalho identifica atributos opcionais e obrigatórios, bem como não requer que um documento JSON possua atributos específicos, como, por exemplo, os atributos *type* e *_id* utilizados na primeira etapa do processo proposto por Ruiz, Morales e Molina (2015). Quanto às etapas do processo, a etapa de extração de objetos JSON foi a que mais contribuiu para o presente trabalho.

3.3 *Schema Management for Document Stores*

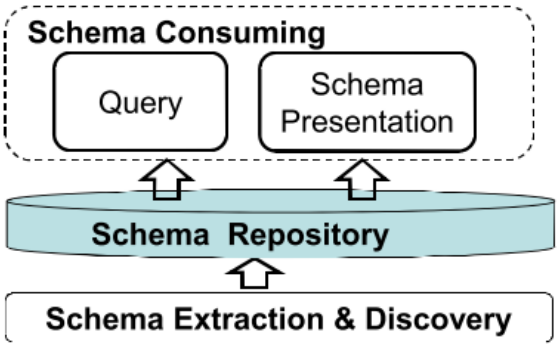
Os autores propõem um *framework*[1] para gerenciamento de esquemas de BDs de documentos (WANG et al., 2015). O *framework* descobre e persiste esquemas de documentos JSON (ver seção 2.4), além de suportar consultas e sumarização de esquemas.

O *framework* proposto possui três componentes (Figura 23): (*i*) componente de extração e descoberta de esquemas; (*ii*) componente de armazenamento de esquema; e,

[1] Um *framework*, em desenvolvimento de *software* é uma abstração que une códigos comuns entre vários projetos de *software*, provendo uma funcionalidade genérica (Wikimedia, 2017).

(*iii*) um componente para consulta e sumarização de esquemas.

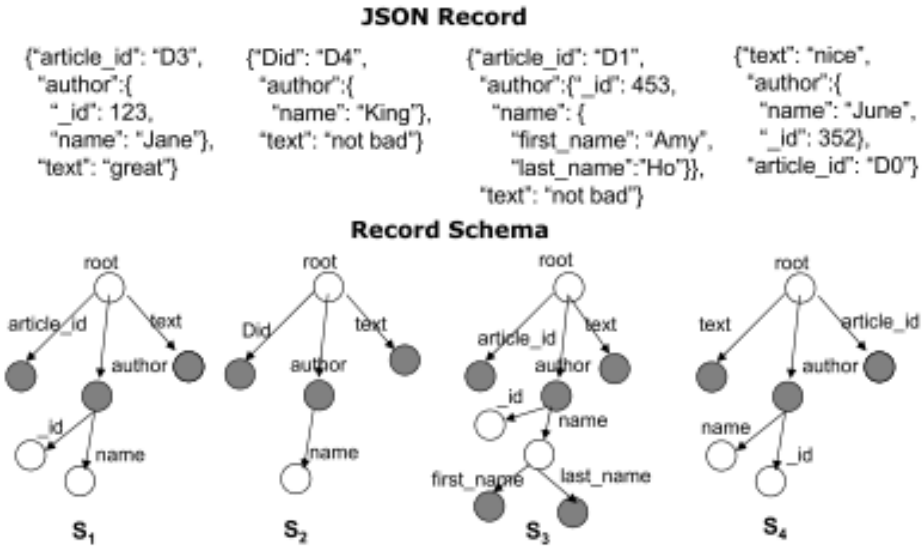
Figura 23: *Framework* para gerenciamento de esquemas proposto por Wang et al. (2015).



Fonte: Wang et al. (2015, p. 923)

O componente de extração e descoberta de esquemas, identifica todos os esquemas de registro distintos, a partir de um *dataset*, agrupando os equivalentes em categorias com base em sua forma canônica. Apenas os nomes dos campos são considerados para montar o esquema de registro (não fazendo referência a tipos de dados). A Figura 24 mostra quatro registros JSON e seus esquemas de registro correspondentes.

Figura 24: Registros JSON e seus esquemas de registro



Fonte: Wang et al. (2015, p. 923)

O componente de armazenamento de esquemas é responsável pela persistência dos esquemas de BD de documentos, suportando eficientemente o processo de extração e de consumo de esquemas. Esse componente utiliza uma estrutura de dados hierárquica,

a *eSiBU-Tree* (*encoded Schema in Bucket Tree*), proposta pelos autores, cujos caminhos persistem categorias de esquemas de registro equivalentes.

O componente de consulta fornece a funcionalidade de encontrar a resposta exata a certos tipos de consultas em esquemas. Duas consultas base são disponibilizadas pelo componente: existência de esquema e existência de atributo.

Por fim, o componente de sumarização de esquema tem como objetivo fornecer uma representação resumida dos esquemas. O conceito de *skeleton* é usado para gerar um esquema único (*core schema*) do objeto, o qual é formado pelos atributos que aparecem com maior frequência nos esquemas dos registros.

Algumas ideias do trabalho proposto por Wang et al. (2015) contribuiriam para o presente trabalho:

- Extração do esquema de cada registro JSON (primeira etapa);
- Utilizar uma estrutura de dados hierárquica (semelhante à *eSiBu-Tree*).

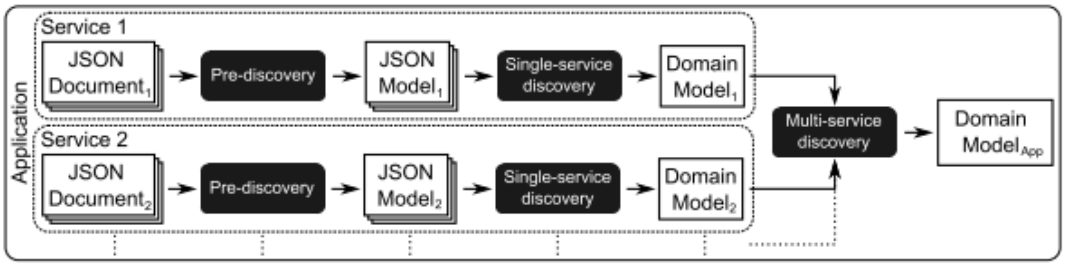
3.4 *Discovering Implicit Schemas in JSON Data*

Os autores propõem uma abordagem para gerar o esquema básico de um conjunto de documentos JSON (ver seção 2.4) obtidos de um *web service* (IZQUIERDO; CABOT, 2013). O processo é composto por três etapas dirigidas por MDE (Figura 25): (*i*) pré-descoberta, (*ii*) descoberta de serviço único e (*iii*) descoberta multi-serviço. É um processo iterativo, no qual novos documentos JSON (das mesmas ou diferentes APIs dentro do *web service*) contribuem para enriquecer o esquema gerado. O esquema ajuda a compreender serviços únicos e inferir possíveis relações entre eles, sugerindo possíveis composições e fornecendo uma visão geral do domínio do aplicativo.

Na primeira etapa, documentos JSON são obtidos a partir de múltiplas chamadas a um *web service*. Nesta etapa, também é realizada a definição da gramática JSON em *Xtext*[2] (utilizado para definição de linguagens de programação) (Figura 26a). O metamodelo da linguagem JSON (Figura 26b) e o injetor de modelo JSON são gerados automaticamente. A Figura 27 ilustra a fase de pré-descoberta, na qual os modelos JSON

[2] https://www.eclipse.org/Xtext/

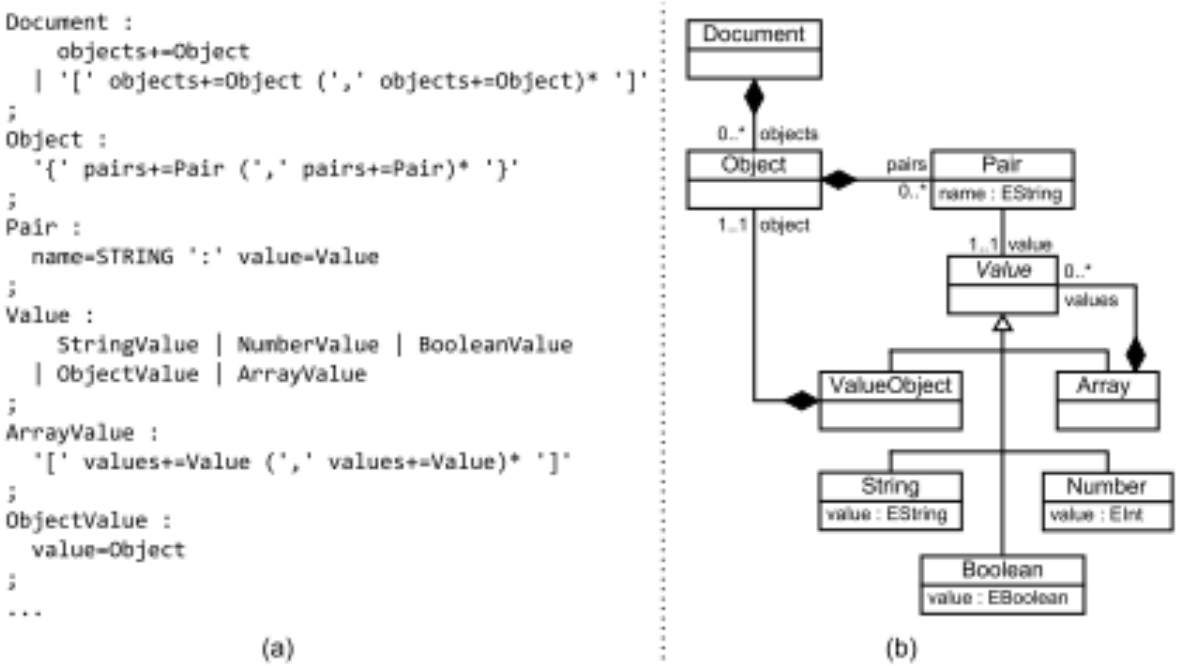
Figura 25: Processo de descoberta de esquemas de documentos JSON proposto por Izquierdo e Cabot (2013).



Fonte: Izquierdo e Cabot (2013, p. 72)

conformes ao metamodelo JSON são injetados a partir de documentos JSON, conforme a gramática JSON.

Figura 26: Gramática e metamodelo JSON em *Xtext*.

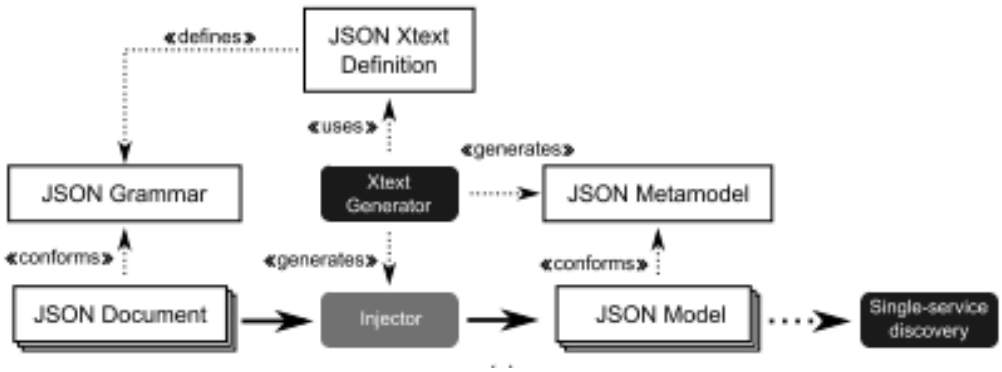


Fonte: Izquierdo e Cabot (2013, p. 75)

A segunda etapa visa a obtenção de informações de esquema para um dado serviço. O processo é executado para cada objeto JSON em dois modos de execução: criação de um novo esquema ou refinamento de um esquema existente. Seu produto é um modelo de domínio para cada serviço.

A terceira etapa é encarregada de compor as informações de esquema obtidas na fase anterior, a fim de obter uma visão geral do domínio da aplicação. É realizado o

Figura 27: Processo de pré-descoberta.



Fonte: Izquierdo e Cabot (2013, p. 75)

merge dos modelos de domínio de serviço obtidos na segunda etapa, produzindo o modelo de domínio da aplicação.

Algumas ideias do trabalho proposto por Izquierdo e Cabot (2013) contribuíram para o presente trabalho:

- Criação de um novo esquema ou refinamento de um esquema existente (segunda etapa);
- *Merge* dos modelos de domínio de serviço (terceira etapa).

No presente trabalho, na etapa de unificação de esquemas de registro, quando uma propriedade possui mais de um tipo de dados distinto, uma operação de união é utilizada, mantendo as variações possíveis. Já, no trabalho proposto por Izquierdo e Cabot (2013), é selecionado o tipo de dados mais genérico em cada caso.

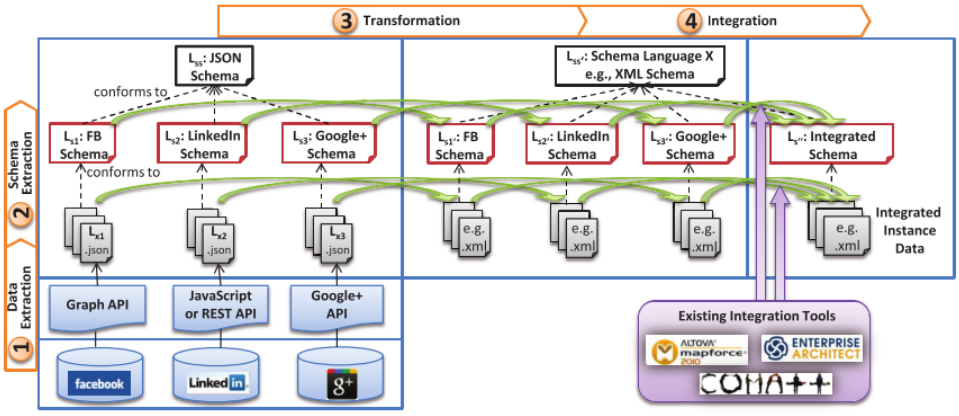
3.5 *User Profile Integration Made Easy — Model-Driven Extraction and Transformation of Social Network Schemas*

Os autores propõem uma abordagem semi-automática para extração e transformação de esquemas de perfis de usuário em redes sociais (*Facebook*[3], *Google+*[4] e *Linkedin*[5]). Uma vez que os usuários são membros de várias redes sociais, os perfis integrados de várias redes são desejados para alcançar uma visão abrangente sobre os usuários. A abordagem

[3] https://www.facebook.com
[4] https://plus.google.com
[5] https://www.linkedin.com

proposta é composta por quatro etapas: (*i*) extração de dados; (*ii*) extração de esquemas; (*iii*) transformação; e, (*iv*) integração (Figura 28).

Figura 28: Abordagem para extração de esquemas proposta por Wischenbart et al. (2012)



Fonte: Wischenbart et al. (2012, p. 940)

Na primeira etapa, dados de instâncias JSON (ver seção 2.4) são extraídos das redes sociais através de suas APIs correspondentes, obtendo assim um conjunto de múltiplas amostras de dados diferentes, cada uma possuindo, eventualmente, fragmentos de dados complementares. Um pequeno exemplo de um usuário do *Facebook* é mostrado no lado esquerdo da Figura 29.

Na segunda etapa, múltiplos esquemas são inferidos dos registros obtidos na etapa anterior. Esses esquemas são expressos na linguagem JSON *Schema* (ver seção 2.5), sendo usados para conceber um único esquema consistente através de uma operação de *merge*, exemplificado no lado direito da Figura 29.

Na terceira etapa, uma transformação dos esquemas JSON extraídos para modelos conceituais (metamodelo ECORE[6]) é apresentada, a fim de permitir a aplicação de ferramentas de integração de última geração.

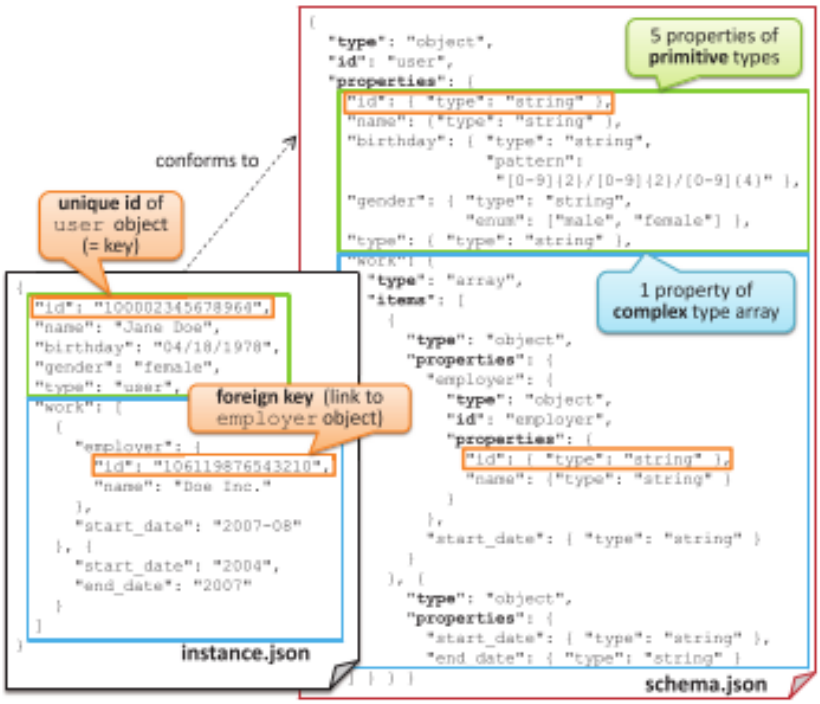
Na quarta etapa são aplicados processos de integração nos esquemas transformados. Isso é feito com o suporte de ferramentas de modelagem de propósito geral (como *Enterprise Architect*[7]), ferramentas para verificação de similaridade (como COMA++[8]) e

[6] ECORE - http://www.eclipse.org/modeling/emf/ - um modelo canônico que possibilita a transformação de esquemas para outros formatos, como XML *Schema*, OWL (*Web Ontology Language*), classe *Java* etc.

[7] http://www.enterprisearchitect.at/

[8] https://dbs.uni-leipzig.de/de/Research/coma.html

Figura 29: JSON e JSON *Schema*



Fonte: Wischenbart et al. (2012, p. 940)

para mapeamento de esquemas (como *MapForce*[9]).

Algumas ideias do trabalho proposto por Wischenbart et al. (2012) contribuíram para o presente trabalho:

- Adições e modificações dos esquemas extraídos anteriormente (segunda etapa);
- Produção der um único esquema coerente, utilizando mesclagem (segunda etapa).

3.6 Comparativo

O artigo de IMHOF, FROZZA e MELLO (2017) intitulado "Um *Survey* sobre Extração de Esquemas de Documentos JSON", apresenta uma análise comparativa entre trabalhos relacionados ao tema de extração de esquemas de documentos JSON. Este artigo foi utilizado como referência para a definição dos principais trabalhos relacionados à ferramenta proposta e facilitou na identificação das características de cada trabalho.

A Tabela 1 apresenta um comparativo das principais características observadas

[9] https://www.altova.com/mapforce

nos trabalhos relacionados. A última coluna se refere a este trabalho, que é também comparado com os demais. As características consideradas foram as seguintes: (*i*) Origem (fonte de origem usada para obter documentos JSON); (*ii*) Objetivo (perspectiva de uso dos esquemas extraídos); (*iii*) Abordagem (abordagem utilizada para obter o esquema final); (*iv*) Formato de entrada suportado (se suporta mais de um formato de entrada); (*v*) Modelo Intermediário ; (*vi*) Modelo de saída (como o esquema é disponibilizado ao final do processo); e (*vii*) Etapas do processo (um resumo das etapas do processo de extração adotado).

Tabela 1: Comparação entre propostas para extração de esquemas de BDs NoSQL

	(KLETTKE; STÖRL; SCHERZINGER, 2015)	(RUIZ; MORALES; MOLINA, 2015)	(WANG et al., 2015)	(IZQUIERDO; CABOT, 2013)	(WISCHENBART et al., 2012)	(Costa, 2017)
Origem dos dados	*Dataset* no *MongoDB*	*MongoDB*, *CouchDB* e *HBase*	*Datasets* JSON	APIs de serviços *web*	APIs de redes sociais	*Dataset* no *MongoDB*
Objetivo	Ferramenta para visualização e validação de esquemas.	Ferramenta para visualização e validação de esquemas	*Framework* para consulta e visualização de esquemas	Criar visão de domínio para os serviços da API	Integrar perfis de usuário em redes sociais	Ferramenta *web* para extração e visualização de esquemas.
Abordagem	Organização hierárquica (grafo)	Transformação de modelos	Organização hierárquica (árvore)	Transformação de modelos	Transformação de modelos	Organização hierárquica e Transformação de modelos
Formato entrada	JSON	JSON	JSON	JSON	JSON	JSON e JSON Estendido
Modelo Interm.	*Structure Identification Graph* (SG)	Metamodelo JSON	*eSiBu-Tree*	Metamodelo JSON (ECORE)	JSON *Schema*	*Raw Schema Unified Structure* (RSUS)
Modelo Saída	JSON *Schema*	Metamodelo de esquema JSON	*eSiBu-Tree*	Modelo de domínio da Aplicação (ECORE)	Modelo de classes (ECORE)	JSON *Schema*
Tipo de software	Algoritmo	Algoritmo	*Framework*	*Plugin Eclipse*	Algoritmo	SaaS (Software-as-a-Service)
Etapas	a) Seleção de documentos JSON; b) Extração da estrutura dos documentos; c) Construção do grafo SG; d) Geração do JSON *Schema*.	a) Extração da estrutura dos dados JSON (*raw schema*) ; b) Transformação para esquemas JSON; c) Transformação para esquema NoSQL.	a) Seleção de documentos JSON; b) Criação de registro na *eSiBu-Tree*; c) Visualização do esquema unificado (*skeleton*).	a) Pré-descoberta de documentos JSON; b) Extração de esquema do serviço; c) Criação do esquema de domínio.	a) Extração de dados; b) Extração de esquemas; c) Transformação; d) Integração.	a) Extração da estrutura dos documentos); b) Agrupação de estruturas extraídas; c) Construção da RSUS; d) Geração do JSON *Schema*.

Fonte: Autor (2017)

Este capítulo apresentou uma análise dos trabalhos similares à ferramenta proposta. Percebe-se que ambos os trabalhos compartilham um objetivo geral em comum: manipulação e gerenciamento de esquemas. No entanto, a ferramenta proposta se destaca das demais por ser uma aplicação *web* e oferecer uma interface gráfica intuitiva para o usuário. A ferramenta também mantém o histórico dos esquemas extraídos e é a única ferramenta para este fim no mercado. Destaque também para o formato de dados de entrada suportado e para o modelo de saída. O presente trabalho, bem como a proposta de Klettke, Störl e Scherzinger (2015), utilizam como modelo de saída uma especificação padrão para definição de esquemas JSON (JSON *Schema* (ver seção 2.5)).

Os diferenciais dessa ferramenta quanto aos trabalhos relacionados são:

- SaaS (*Software-as-a-Service*) — acesso via qualquer *browser*;
- Persistência dos esquemas gerados;
- Histórico de esquemas gerados;
- Visualização e *download* de esquemas gerados;
- Suporte a JSON Estendido.

4 *JSONSchema Discovery*

A partir do estudo realizado sobre os trabalhos relacionados, no qual foram apresentados alguns exemplos de processos de extração de esquemas (ver seção 3), a ferramenta *JSONSchema Discovery* foi projetada. Ao longo dessa seção são apresentadas informações referentes ao projeto (seção 4.1), as etapas do processo de extração (seção 4.2) e a implementação da ferramenta proposta (seção 4.3).

4.1 Projeto

JSONSchema Discovery é uma aplicação *web* para extração de esquemas – no formato padrão JSON *Schema* - de uma coleção de documentos JSON armazenados em um BD de documentos. A aplicação mantém o histórico de esquemas extraídos para cada usuário e o mesmo pode visualizar ou baixar o esquema extraído.

O projeto da ferramenta iniciou pelo levantamento de requisitos, os quais representam a descrição das necessidades dos usuários perante um *software*. O objetivo desta fase é identificar e documentar os requisitos. É necessário que a documentação dos requisitos seja feita de forma clara e que comunique efetivamente a informação que o usuário busca apresentar (LARMAN, 2004).

Os requisitos de um *software* podem ser classificados em Requisitos Funcionais (RF) ou Requisitos Não-Funcionais (RNF). RFs são os responsáveis por definir funções e comportamentos do *software*. Já RNF são os responsáveis por especificar as qualidades do *software*, isto é, a confiabilidade, a segurança e o desempenho.

Para o levantamento de RFs foram levadas em consideração características de aplicações SaaS e funcionalidades que deveriam existir na ferramenta. Os RNFs foram definidos a partir de estudos de tecnologias *web* modernas. A seguir são sumarizados os requisitos levantados. Informações mais detalhadas podem ser consultadas no Apêndice A.

Os requisitos funcionais da ferramenta são os seguintes:

- RF01: Criar conta;
- RF02: Entrar;
- RF03: Sair;
- RF04: Extrair esquema;

- RF05: Deletar resultado;
- RF06: Visualizar resultado;
- RF07: Baixar resultado;
- RF08: Listar os resultados;
- RF09: Notificação de resultado.

Os requisitos não-funcionais da ferramenta são os seguintes:

- RNF01: Arquitetura *client-server*;
- RNF02: Utilizar linguagem de programação *TypeScript*;
- RNF03: Utilizar o *framework Angular* e a biblioteca *Angular Material* no *front-end*;
- RNF04: Utilizar o *framework Node.js* no *back-end*;
- RNF05: Utilizar *MongoDB* como SGBD.

Todas as funcionalidades da ferramenta podem ser visualizadas através de um Diagrama de Casos de Uso (Figura 30). Segundo Silva (2009, p. 35):

> Um diagrama de casos de uso é um diagrama de modelagem dinâmica de sistema em alto nível de abstração, que modela as funcionalidades do *software* (casos de uso) com os elementos externos que interagem com o *software* (atores).

Figura 30: *JSONSchema Discovery* - Diagrama de casos de uso

Fonte: Autor (2017)

4.2 Processo de extração de esquemas

A extração de esquemas de BDs de documentos envolve a descoberta de documentos e os atributos de cada documento. Sendo assim, o algoritmo proposto neste trabalho percorre todos os documentos JSON armazenados e analisa suas propriedades para identificar o esquema bruto de cada documento e, então, unificar os esquemas brutos e gerar um JSON *Schema* único que represente a coleção de documentos como um todo.

Para resumir a informação estrutural de todos os esquemas brutos persistidos é definida uma nova estrutura de dados, na qual são armazenadas as propriedades dos esquemas brutos, como objetos aninhados, *arrays* ou tipos de dados JSON primitivos (ver seção 2.4) ou JSON Estendido (ver seção 2.4.1). A estrutura de dados e sua construção é descrita na seção 4.2.3 em detalhes. A seguir são descritos os processos de obtenção dos esquemas brutos dos documentos e de agrupamento de esquemas brutos idênticos.

4.2.1 Obtendo os esquemas brutos dos documentos

O primeiro passo para extrair o JSON *Schema* é obter o esquema bruto (*raw schema*) de cada documento JSON de uma coleção, passo semelhante ao apresentado por Ruiz, Morales e Molina (2015) (ver seção 3.2), e armazenar os resultados em uma coleção temporária do *MongoDB*. O esquema bruto contém a mesma estrutura que o documento JSON original com relação à campos, objetos aninhados e *arrays*. Cada valor primitivo no documento é substituído no esquema bruto por seu tipo de dado JSON (por exemplo, *string* ou *number*). Um diferencial nesta proposta é o fato de considerar os tipos de dados presentes no JSON Estendido do *MongoDB* (por exemplo: *date*, *objectId*, entre outros (ver seção 2.4.1)).

O Algoritmo 1 realiza a extração do esquema bruto de cada documento, dada uma coleção de documentos JSON como entrada. Como saída, é retornada uma coleção de esquemas brutos.

O Algoritmo 2 descreve a função *BuildRawSchema* invocada no Algoritmo 1. Ela recebe um valor como entrada e realiza a extração do esquema bruto do valor. Se o tipo do valor for um tipo JSON primitivo ou um tipo JSON Estendido, o algoritmo retorna o nome do tipo. Caso contrário, é aplicada recursão para valores do tipo objeto ou *array*.

Algoritmo 1: Operação de extração do *raw schema* dos documentos

```
1  function Parser(collection);
   Input  : collection of documents JSON
   Output : rawSchemes of documents JSON
2  begin
3      rawSchemes ← ∅;
4      for document ∈ collection do
5          documentRawSchema ← {};
6          for key ∈ keys(document) do
7              value ← document[key];
8              documentRawSchema[key] ← BuildRawSchema(value);
9          end
10         add documentRawSchema to rawSchemes;
11     end
12     return rawSchemes;
13 end
```

4.2.2 Agrupando esquemas brutos iguais

Na segunda etapa, duas operações de agregação são aplicadas para extrair uma coleção de objetos JSON únicos , ou seja, o número mínimo de objetos necessários para executar o processo de unificação dos esquemas brutos. Operações de agregação são pertinentes à maioria dos BDs e representam uma vantagem no desempenho (comparando com operações *mapreduce*), pois é um método de processamento nativo quando um algoritmo tem que lidar com todos os objetos em um BD.

A primeira operação de agregação é aplicada na coleção resultante da primeira etapa com o objetivo de agrupar os documentos que possuem o mesmo esquema bruto. O resultado dessa operação é armazenado em uma coleção temporária que é utilizada pela segunda operação de agregação.

Antes de aplicar a segunda operação de agregação, é realizada a ordenação alfabética das propriedades dos esquemas brutos resultantes da primeira etapa de agregação. Considerando que os elementos de um documento JSON não precisam seguir a mesma ordem que estão definidos no esquema, o objetivo desta ordenação é reduzir ainda mais a quantidade de documentos com esquemas brutos distintos. Após a ordenação ser concluída, é iniciada a segunda operação de agregação que resulta na menor quantidade de documentos com esquema bruto distintos.

Algoritmo 2: Composição do *raw schema*

```
1  function BuildRawSchema(value);
   Input  : value of key-value pair
   Output : rawSchema of value
2  begin
3      instance ← NULL;
4      if extendedJSONTypeOf(value) ≠ NULL then
5          instance ← extendedJSONTypeOf(value);
6      else if value = UNDEFINED then
7          instance ← "undefined";
8      else if value = NULL then
9          instance ← "null";
10     else if typeof(value) = Array then
11         instance ← ∅;
12         for item ∈ value do
13             itemRawSchema ← BuildRawSchema(item);
14             add itemRawSchema to instance;
15         end
16     else if typeof(value) = Object then
17         instance ← {};
18         for key ∈ keys(value) do
19             keyValue ← value[key];
20             instance[key] ← BuildRawSchema(keyValue);
21         end
22     else
23         instance ← typeof(value);
24     end
25     return instance;
26 end
```

4.2.3 Unificação de Esquemas Brutos

Para a unificação de esquemas brutos foi necessário definir uma estrutura de dados temporária, que armazenasse de forma performática, informações sobre a estrutura hierárquica de cada esquema bruto, sendo essas informações: os atributos dos objeto e seus tipos de dados, os itens dos *arrays* e seus tipos de dados, caminho da raiz do documento até uma propriedade ou item qualquer, bem como a quantidade de ocorrências para cada propriedade, com base no caminho e no tipo de dado, nos documentos de uma coleção. Assim, as informações de todos os esquemas brutos são armazenados na chamada Estrutura Unificada de Esquema Bruto ou, em Inglês, ***R**aw **S**chema **U**nified **S**tructure* (RSUS). A RSUS é um documento JSON que possui cinco tipos de propriedades:

"field" - Um objeto JSON, que representa um atributo do documento, contendo os atributos *"name"*, *"path"*, *"count"* e *"types"*, sendo:

- *"name"* - nome do atributo;
- *"path"* - informação do caminho até o atributo, iniciando pela raiz do documento;
- *"count"* - número de ocorrências desse atributo nos documentos;
- *"types"* - um *array* contendo os tipos de dados que ocorreram para esse atributo nos documentos. Um tipo de dado pode ser *"primitiveType"*, *"extendedType"*, *"objectType"* ou *"arrayType"*.

"primitiveType" - Um objeto JSON que representa um tipo de dado JSON primitivo, contendo os atributos *"name"*, *"path"* e *"count"*, sendo:

- *"name"* - nome do tipo de dado, podendo ser: *"String"*, *"Number"*, *"Boolean"* ou *"Null"*;
- *"path"* - informação do caminho até o tipo de dado iniciando pela raiz do documento;
- *"count"* - número de ocorrências desse tipo de dado em um atributo nos documentos.

"extendedType" - Um objeto JSON que representa um tipo de dado JSON Estendido, contendo os atributos *"name"*, *"path"* e *"count"*, sendo:

- *"name"* - nome do tipo de dado, podendo ser um dos tipos de dados JSON Estendido listados na seção 2.4.1;
- *"path"* - informação do caminho até o tipo de dado iniciando pela raiz do documento;
- *"count"* - número de ocorrências desse tipo de dado em um atributo nos documentos.

"objectType" - Um objeto JSON que representa um objeto, contendo os atributos *"name"*, *"path"*, *"count"* e *"fields"*, sendo:

- *"name"* - atributo com o valor *"Object"*;
- *"path"* - informação do caminho até o tipo de dado iniciando pela raiz do documento;
- *"count"* - número de ocorrências desse tipo de dado em um atributo nos documentos;

- *"fields"* - um *array* contendo os atributos que ocorreram para este objeto nos documentos extraídos. Um atributo é uma propriedade do tipo *"field"* explicada anteriormente.

"arrayType" - Um objeto JSON que representa um *array*, contendo os atributos *"name"*, *"path"*, *"count"* e *"items"*, sendo:

- *"name"* - atributo com o valor *"Array"*;
- *"path"* - informação do caminho até o tipo de dado iniciando pela raiz do documento;
- *"count"* - número de ocorrências desse tipo de dado em um atributo nos documentos;
- *"items"* - um *array* contendo os elementos que ocorreram para este *array* nos documentos extraídos. Um elemento pode ser uma propriedade do tipo *"objectType"*, *"primitiveType"*, *"extendedType"* ou *"arrayType"*.

Na terceira etapa ocorre a unificação dos esquemas brutos resultantes da etapa 2 (seção 4.2.2). Para isto, é realizada a construção da RSUS a partir dos esquemas brutos extraídos. As propriedades de cada esquema bruto são percorridas em *preorder* e, para cada propriedade, é realizada uma adição ou extensão de uma propriedade na RSUS. Caso a propriedade represente um atributo de objeto, então é realizado uma adição ou extensão de uma propriedade *field* na RSUS. Caso o valor da propriedade represente um tipo JSON ou JSON Estendido, então é realizado uma adição ou extensão de uma propriedade *type* (*primitiveType*, *extendedType*, *objectType* ou *arrayType*) na RSUS. Por fim, caso a propriedade represente um elemento de um *array*, então é realizado a adição ou extensão de um *item* na RSUS, sendo *item* uma propriedade *primitiveType*, *extendedType*, *objectType* ou *arrayType*. Toda adição ou extensão, leva em conta o caminho da raiz do esquema bruto até a propriedade que está sendo analisada.

Na próxima seção é descrito o mapeamento da estrutura RSUS para JSON *Schema*, que é o resultado final do processo de extração de esquemas proposto.

4.2.4 Obtendo o JSON *Schema*

Uma vez realizada a unificação dos esquemas brutos, um algoritmo baseado em transformação de modelos é empregado, para transformar a estrutura RSUS em um JSON

Schema válido. A ideia geral dessa etapa é gerar um JSON *Schema* para cada propriedade na RSUS, respeitando a estrutura hierárquica do esquema bruto.

O processo de mapeamento inicia pelo Algoritmo 3, no qual cada *field* é mapeado para um JSON *Schema*. O Algoritmo 5 verifica qual o tipo RSUS do valor e obtêm o respectivo JSON *Schema* para o tipo. Os algoritmos 4, 6, 7, 8, e 9 mapeiam uma propriedade RSUS, informada na entrada, para um JSON *Schema*.

Algoritmo 3: Mapeamento RSUS → JSON *Schema*

```
1 function getFieldsSchema(fields);
  Input  : fields
  Output : properties
2 begin
3 |   properties ← {};
4 |   for field ∈ fields do
5 |   |   properties[field.name] ← getSchemaFromValue(field);
6 |   end
7 |   return properties;
8 end
```

Algoritmo 4: RSUS extendedType → JSON *Schema*

```
1 function getExtendedTypeSchema(type);
  Input  : type
  Output : schema
2 begin
3 |   schema ← {};
4 |   schema["$ref"] ←" #/definitions/type.name";
5 |   schema["count"] ← type.count;
6 |   return schema;
7 end
```

Algoritmo 5: Mapeanto RSUS → JSON *Schema*

```
1  function getSchemaFromValue(value);
   Input  : value
   Output : schema
2  begin
3      schema ← NULL;
4      if ¬isEmpty(value.types) then
5          schema ← getFieldSchema(value);
6      else
7          if isExtendedType(value.name) then
8              schema ← getExtendedTypeSchema(value.name, value.count);
9          else if isPrimitiveType(type.name) then
10             schema ← getPrimitiveTypeSchema(NULL, value.name, value.count);
11         else if isArray(value.name) then
12             schema ← getArrayTypeSchema(value);
13         else
14             schema ← getObjectTypeSchema(value);
15         end
16     end
17     return schema;
18 end
```

Algoritmo 6: RSUS objectType → JSON *Schema*

```
1  function getObjectTypeSchema(object);
   Input  : object
   Output : schema
2  begin
3      properties ← getFieldsSchema(object.fields, object.count);
4      schema ← {};
5      schema["type"] ←" object";
6      schema["properties"] ← properties;
7      schema["additionalProperties"] ← false;
8      schema["count"] ← object.count;
9      schema["required"] ←
         getRequiredFieldNamesFromObject(schema.properties, schema.count);
10     return schema;
11 end
```

Algoritmo 7: RSUS primitiveType → JSON *Schema*

```
1  function getPrimitiveTypeSchema(name, type, count);
   Input  : name
   Input  : type
   Input  : count
   Output : schema
2  begin
3      schema ← {};
4      if name ≠ NULL then
5          schema["name"] ← name;
6      end
7      schema["type"] ← type;
8      schema["count"] ← count;
9      return schema;
10 end
```

Algoritmo 8: RSUS arrayType → JSON *Schema*

```
1  function getArrayTypeSchema(array);
   Input  : array
   Output : schema
2  begin
3      items ← NULL;
4      if sizeOf(array.items) = 1 then
5          item ← array.items.pop();
6          itemSchema ← getSchemaFromValue(item);
7          items ← itemSchema;
8      else
9          items ← {};
10         items["anyOf"] ← ∅;
11         for item ∈ array.items do
12             itemSchema ← getSchemaFromValue(item);
13             add itemSchema in items["anyOf"];
14         end
15     end
16     schema ← {};
17     schema["name"] ← array.path;
18     schema["type"] ←" array";
19     schema["items"] ← items;
20     schema["additionalItems"] ← true;
21     schema["count"] ← array.count;
22     schema["minItems"] ←
         getMinimumSizeOfList(schema.items, schema.count);
23     return schema;
24 end
```

Algoritmo 9: RSUS field → JSON *Schema*

```
1  function getFieldSchema(field);
   Input  : field
   Output : schema
2  begin
3      schema ← {};
4      if sizeOf(field.types) = 1 then
5          type ← field.types.pop();
6          if isExtendedType(type.name) then
7              schema ← getExtendedTypeSchema(type.name, type.count);
8          else if isPrimitiveType(type.name) then
9              schema ←
                getPrimitiveTypeSchema(field.name, type.name, field.count);
10         else if isArray(type.name) then
11             schema ← getSchemaFromValue(type);
12         else
13             schema ← getSchemaFromValue(type);
14             schema["name"] ← type.path;
15         end
16     else
17         types ← ∅;
18         for type ∈ field.types do
19             typeSchema ← getSchemaFromValue(type);
20             add typeSchema in types;
21         end
22         schema ← {};
23         schema["anyOf"] ← types;
24         schema["count"] ← field.count;
25     end
26     return schema;
27 end
```

4.3 Implementação

4.3.1 Tecnologias e ferramentas

Esta seção descreve as principais tecnologias e ferramentas empregadas na implementação do *JSONSchema Discovery*.

4.3.1.1 *Javascript*

JavaScript[1] é uma linguagem de programação originalmente desenvolvida para a execução de código dinâmico em páginas *web*. Ela foi adotada por praticamente todos os grandes navegadores do mercado, como *Google Chrome*[2], *Firefox*[3], *Safari*[4] e *Opera*[5], tornando-a uma das linguagens mais populares do mundo (CROCKFORD, 2008). Ela é uma linguagem interpretada, com tipagem fraca e dinâmica, bem como suporte à programação funcional e à programação orientada a objetos.

Inicialmente utilizada apenas como *script* para programar o comportamento de páginas e manipular os seus elementos, a linguagem *JavaScript* evoluiu a tal ponto que é utilizada como linguagem de consulta em BDs, como o *MongoDB* e o *CouchDB*; em *frameworks* de manipulação de DOM, como o *Angular*, e em plataformas de desenvolvimento *server-side* como o *Node.js*.

Devido a sua flexibilidade, eficiência e robustez, *JavaScript* foi escolhida como linguagem de implementação para este trabalho, tanto na parte do servidor — no qual ocorre o processamento de dados e interação com o BD —, quanto na parte do cliente — na qual existe a interação do usuário com páginas HTML, coordenando o comportamento da aplicação. Como toda a ferramenta foi desenvolvida com *JavaScript*, diversas bibliotecas foram necessárias para facilitar o processo.

4.3.1.2 *Typescript*

TypeScript[6] é uma linguagem para desenvolvimento *JavaScript* em larga escala. Com *TypeScript* é possível escrever código utilizando uma estrutura fortemente tipada

1 https://www.javascript.com/
2 https://www.google.com/chrome
3 https://www.mozilla.org/firefox/
4 https://www.apple.com/safari/
5 http://www.opera.com
6 https://www.typescriptlang.org/

e ter este código compilado para *JavaScript* puro. Também é possível identificar erros durante a compilação.

4.3.1.3 *Angular*

Angular[7] é um *framework JavaScript* criado para aumentar a produtividade de desenvolvimento e organização de código no lado do cliente. O *framework* propõe o controle do DOM (*Document Object Model*) de forma declarativa, a extensão do vocabulário HTML, a criação de componentes para o reuso de código e uma forma mais organizada de manter o código *JavaScript*. As principais virtudes da biblioteca são leveza e capacidade de propiciar organização e estrutura para aplicações de grande porte. Por isso, ela foi usada para organizar a parte do cliente no padrão MVC (ver seção 4.3.2), aumentando a velocidade de desenvolvimento, organização e manutenção.

4.3.1.4 *Angular Material*

O *Angular Material*[8] é um conjunto de componentes de interface de usuário, desenvolvidos com o *framework Angular*. Esses componentes seguem as especificações estabelecidas pelo *Material Design*[9] da *Google*.

4.3.1.5 *Node.js*

Node.js[10] é um *framework* orientado a eventos que possibilita a execução de *JavaScript* no lado do servidor. Ele foi desenvolvido utilizando o compilador de interpretação do navegador *Google Chrome*, cujo propósito é acelerar o desempenho de aplicações compilando o código *JavaScript* para o formato nativo de máquina antes de executá-lo.

A principal diferença entre *Node.js* e outros *frameworks web* é a forma como as requisições são tratadas. Enquanto *frameworks* tradicionais utilizam *threads* em paralelo, fazendo com que a alocação de recursos possa ficar bloqueada esperando um evento concorrente finalizar, *Node.js* propõe um conceito de fila de eventos, na qual requisições são tratadas de maneira assíncrona e, portanto, não bloqueante. O *framework* conta com

[7] https://angular.io/
[8] https://material.angular.io/
[9] https://material.io/
[10] https://nodejs.org/en/

milhares[11] de extensões disponíveis no repositório NPM[12], desenvolvidas pela comunidade de *software*, que aceleram o processo de desenvolvimento, uma vez que é possível reutilizar esses componentes.

4.3.1.6 REST

REST (*Representational State Transfer*) é um estilo arquitetural de *software* voltado para sistemas de hipermídia distribuídos, tais como os existentes na WWW (*World Wide Web*). Esse estilo arquitetural, proposto por Fielding (2000), tem sido largamente usado para guiar o *design* e o desenvolvimento de diversas aplicações da *web* moderna. O estilo arquitetural REST enfatiza a escalabilidade das interações entre componentes, a generalidade das interfaces, o desenvolvimento independente de componentes e a utilização de componentes intermediários para reduzir a latência das interações, reforçar a segurança e encapsular sistemas legados (FIELDING, 2000).

4.3.1.7 HTML e CSS

HTML (*HyperText Markup Language*)[13] é uma linguagem de marcação utilizada para produzir páginas na *web* independentemente de plataforma, ou seja, o mesmo código HTML é interpretado da mesma forma por diferentes *browsers*. Atualmente, a HTML se encontra em sua quinta versão, com facilidades para manipulação de dados via *JavaScript* e CSS (*Cascading Style Sheets*).

CSS (*Cascading Style Sheets*)[14] é uma linguagem de folha de estilo, utilizada para descrever a aparência de linguagens de marcação como HTML e XML, incluindo cores, tamanhos, fontes e posicionamento dos elementos na tela.

4.3.2 Arquitetura

A arquitetura utilizada para estruturar a ferramenta *JSONSchema Discovery* é inspirada no modelo *client-server* e utiliza o padrão de projetos MVC (*Model, View, Controller*). Segundo Tanenbaum e Wetherall (2010), no modelo cliente-servidor a comunicação toma a forma do processo cliente, enviando uma mensagem pela rede ao processo servidor.

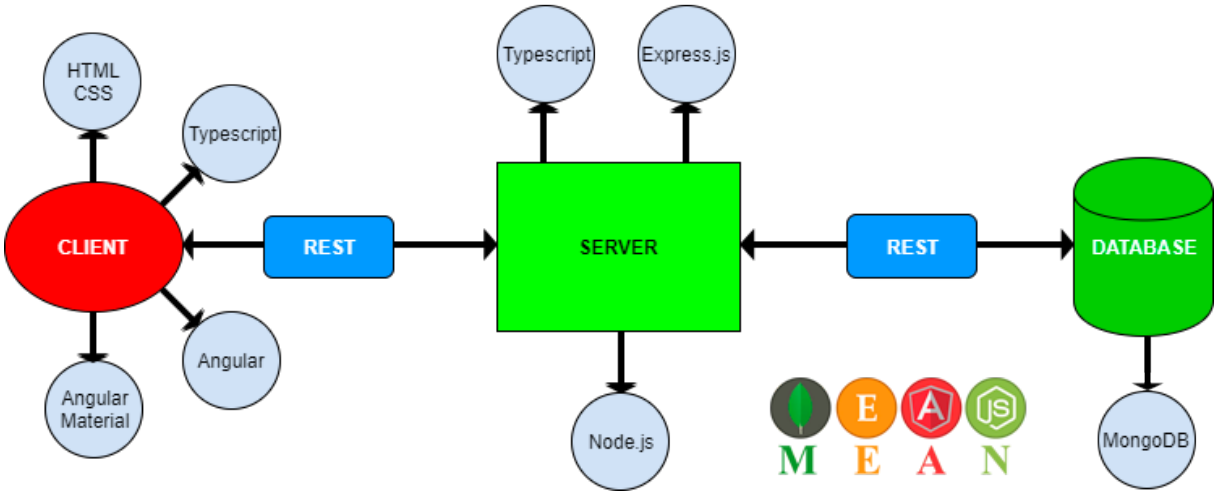
[11] Em 18/10/2017 eram 531.620, com uma taxa de 499 novos *plugins* por dia. http://www.modulecounts.com/
[12] https://www.npmjs.com
[13] https://www.w3.org/TR/html/
[14] https://www.w3.org/Style/CSS/

Quando o servidor recebe a solicitação, ele executa o trabalho solicitado ou procura pelos dados solicitados e envia de volta uma resposta. Ainda, segundo Tanenbaum e Wetherall (2010), esse modelo é amplamente usado no contexto de aplicações *web*.

O MVC, padrão de projetos mais utilizado em arquitetura em desenvolvimento de *software*, determina que a aplicação esteja dividida em três camadas com especialidades distintas. O *Model* é responsável por representar os dados da aplicação, realizando o acesso ao sistema de armazenamento e lógica sobre eles. O *View* trata de como o modelo é apresentado ao usuário. Por fim, o *Controller* recebe todas as solicitações da aplicação e é responsável por executá-las sobre o *Model* e a *View* (FOWLER, 2002). O MVC foi utilizado tanto na parte cliente quanto na parte servidor.

A Figura 31 apresenta uma visão geral da arquitetura da ferramenta proposta. Toda a comunicação entre os componentes se dá por meio do protocolo REST (ver seção 4.3.1.6). As seções 4.3.2.1 e 4.3.2.2 descrevem as responsabilidades dos componentes apresentados nesta visão geral de arquitetura.

Figura 31: Arquitetura da ferramenta proposta.



Fonte: Autor (2017)

4.3.2.1 Cliente

A parte cliente foi inteiramente desenvolvida em HTML, CSS (ver seção 4.3.1.7) e *TypeScript* (ver seção 4.3.1.2), baseado no conceito de componentes. Para a renderização do conteúdo foi utilizado o *framework Angular* (ver seção 4.3.1.3), bem como os componentes padrões da biblioteca *Angular Material* (ver seção 4.3.1.6).

4.3.2.2 Servidor

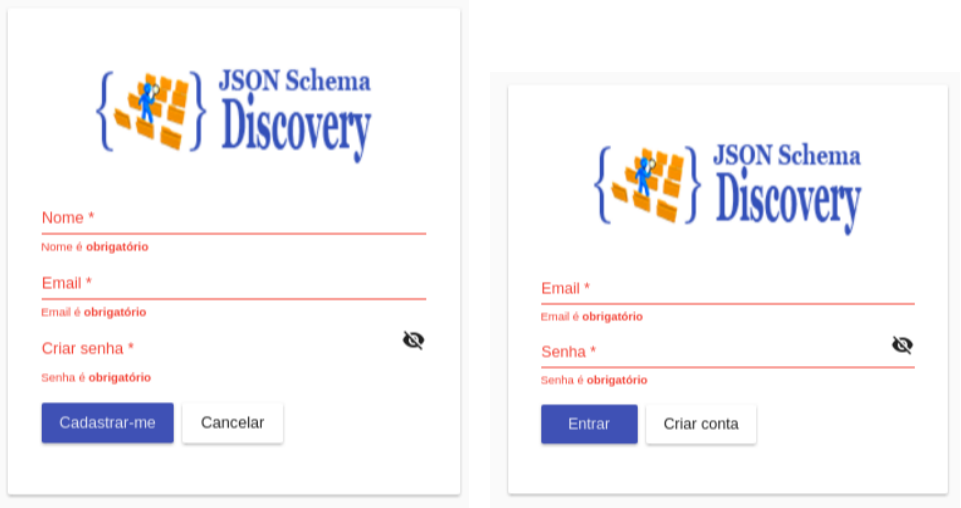
O servidor foi desenvolvido em *Node.js* (ver seção 4.3.1.5) no esquema de *web service* REST. Ele pode ser facilmente instalado localmente em qualquer sistema operacional ou em serviços de hospedagem como *Heroku*[15], *Google Appengine*[16], *Red Hat OpenShift*[17], *WeDeploy*[18], entre outros. A parte de servidor da ferramenta *JSONSchema Discovery* é responsável por todo o controle da lógica da ferramenta. Para persistência de dados é utilizado o SGBD NoSQL orientado a documentos *MongoDB* (ver seção 2.3.5.1).

4.3.3 Funcionalidades

4.3.3.1 Criação de conta

Para acessar a ferramenta *JSONSchema Discovery* é necessário criar uma conta. A Figura 32a exibe o formulário de criação de conta. Os únicos requisitos para esta etapa são que o *e-mail* seja válido e que esse *e-mail* não esteja previamente cadastrado no sistema. Caso algum desses requisitos não sejam cumpridos, uma mensagem de erro é apresentada. Após o cadastro, é realizado o *login* automaticamente.

Figura 32: *JSONSchema Discovery* - Formulários de criação de conta e *login*.



(a) Formulário de criação de conta (b) Formulário de *login*

Fonte: Autor (2017)

15 https://devcenter.heroku.com/articles/getting-started-with-nodejs#introduction
16 https://cloud.google.com/nodejs/
17 https://developers.openshift.com/languages/nodejs/getting-started.html
18 https://wedeploy.com/docs/deploy/deploying-nodejs/

4.3.3.2 *Login*

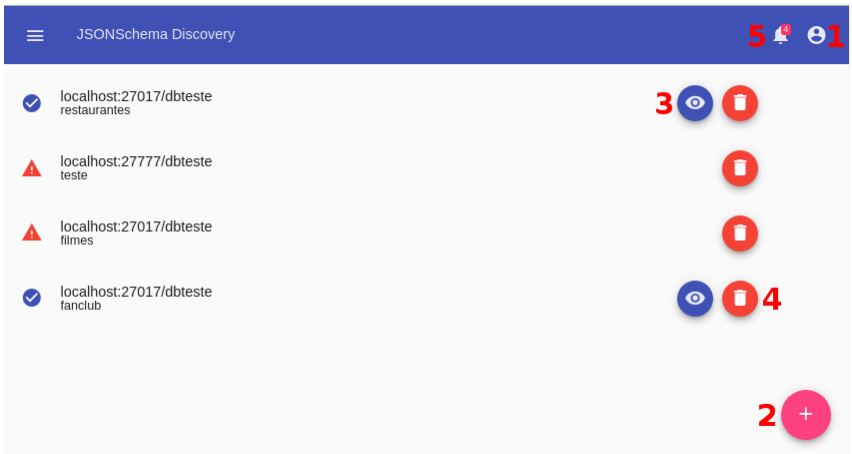
Caso o usuário já tenha cadastro, o mesmo pode acessar a ferramenta. Para isto, o usuário deve informar um *e-mail* e senha válidos. Caso o usuário forneça um *e-mail* que seja inválido ou inexistente, ou a senha não esteja relacionada ao *e-mail* informado, uma mensagem de erro é apresentada. A Figura 32b demonstra o formulário de *login*.

4.3.3.3 Lista de esquemas

Após o *login*, a primeira tela apresentada pela ferramenta é uma lista de esquemas extraídos. A Figura 33 apresenta uma captura de tela já com alguns esquemas extraídos. Nela estão marcadas ações que o usuário pode realizar:

1. Acessar a área do usuário, na qual é possível visualizar informações do perfil do usuário ou sair do sistema;
2. Extrair um novo esquema;
3. Visualizar um esquema já existente;
4. Excluir um esquema já existente;
5. Acessar a área de alertas, na qual é possível tratar as notificações de resultados de extração de esquemas.

Figura 33: *JSONSchema Discovery* - Tela principal.

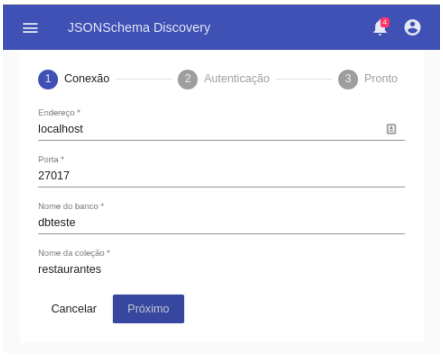


Fonte: Autor (2017)

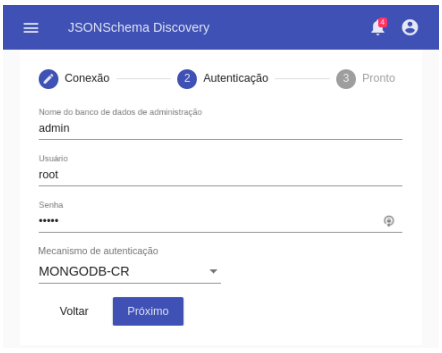
4.3.3.4 Extrair novo esquema

Ao clicar em 'Extrair novo esquema', uma janela com passo-a-passo é apresentada ao usuário (Figura 34). No passo 'Conexão' (Figura 34a), o usuário deve preencher os campos 'Endereço', 'Porta', 'Nome do banco' e 'Nome da coleção'. No próximo passo, 'Autenticação' (Figura 34b), o usuário pode preencher os campos 'Nome do banco de dados de administração', 'Usuário', 'Senha' e escolher um mecanismo de autenticação entre os listados. No passo 'Pronto' (Figura 34c), o usuário pode clicar em 'Extrair JSON *Schema*' ou voltar para os passos anteriores. Os campos do passo 'Conexão' (Figura 34a) são obrigatórios e devem ser preenchidos. Ao clicar em 'Extrair JSON *Schema*' (Figura 34c) o processo de extração de esquema é iniciado e o usuário recebe um alerta informando o resultado da extração.

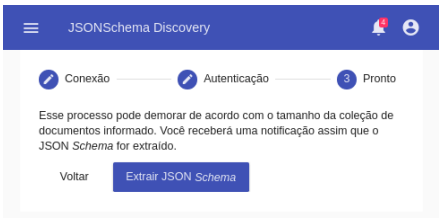
Figura 34: *JSONSchema Discovery* - Passo-a-passo para iniciar uma nova extração de esquema.



(a) Informações de conexão



(b) Informações de autenticação



(c) Confirmação

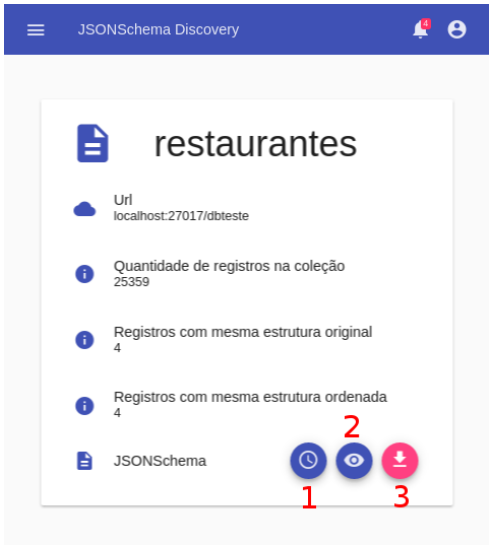
Fonte: Autor (2017)

4.3.3.5 Visualizar esquema

Ao clicar em 'Visualizar esquema', uma janela com os detalhes sobre a extração do esquema é apresentada ao usuário (Figura 35). Nesta tela, o usuário pode fazer as seguintes ações:

1. Visualizar informações detalhadas sobre o tempo decorrido no processo de extração;
2. Visualizar o JSON *Schema*;
3. Baixar o JSON *Schema*.

Figura 35: *JSONSchema Discovery* - Resultado da extração.



Fonte: Autor (2017)

Ao clicar na primeira ação, uma janela com os detalhes sobre o tempo decorrido no processo de extração do esquema é apresentada ao usuário (Figura 36a). Ao clicar na segunda ação, uma janela com o JSON *Schema* gerado é apresentada ao usuário (Figura 36b).

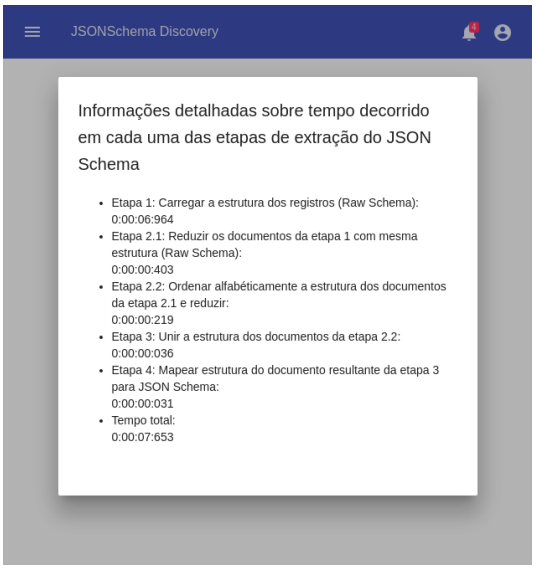
4.3.3.6 Lista de alertas

O usuário recebe uma notificação caso algum problema ocorra durante o processo de extração ou caso o processo finalize com sucesso. Três tipos de notificações podem ser geradas durante o processo de extração de esquemas:

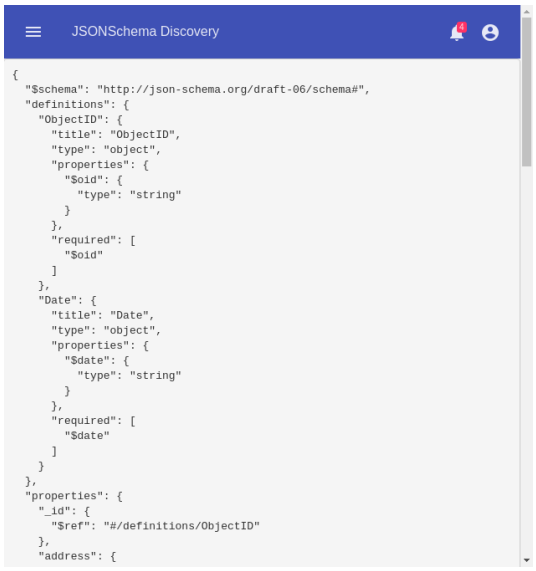
- Erro na conexão com o banco de dados;
- Não há documentos na coleção informada;
- Esquema extraído com sucesso.

Essas notificações são visualizadas ao clicar no ícone de alerta na tela principal, que redireciona para a tela de alertas (Figura 37).

Figura 36: *JSONSchema Discovery* - Detalhes sobre a extração e visualização do JSON *Schema*



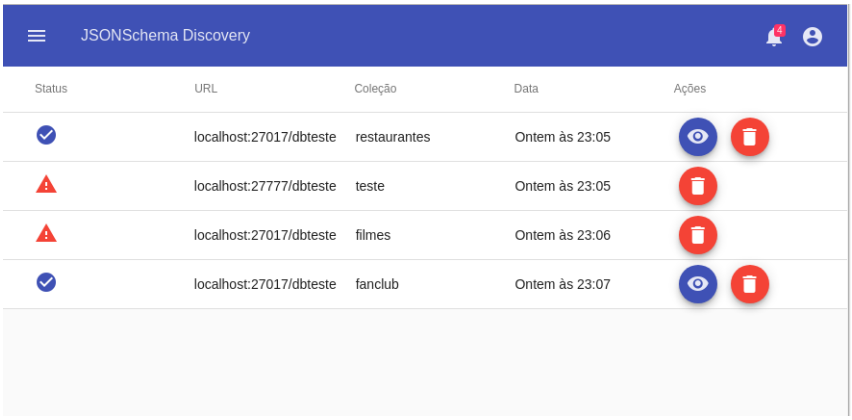
(a) Detalhes de tempo decorrido no processo de extração



(b) Visualização do JSON *Schema*

Fonte: Autor (2017)

Figura 37: *JSONSchema Discovery* - Lista de alertas



Fonte: Autor (2017)

Este capítulo apresentou os requisitos levantados para o projeto e os casos de uso, as etapas do processo de extração, as tecnologias e bibliotecas utilizadas no desenvolvimento da ferramenta, bem como a arquitetura da ferramenta e suas funcionalidades. Os *links* para a ferramenta podem ser encontrados no repositório *GitHub*[19] da aplicação, no qual é possível encontrar também todo o código fonte, fazer sugestões e até colaborar com novas implementações.

19 https://github.com/feekosta/JSONSchemaDiscovery

5 Resultados alcançados

Este capítulo apresenta os resultados obtidos e a avaliação da ferramenta *JSONSchema Discovery*. A avaliação considera três cenários: (A) *qualitativo* — a fim de verificar a qualidade dos mapeamentos JSON →JSON *Schema*; (B) *tempo de processamento* — a fim de analisar o tempo de processamento da ferramenta; (C) *comparação com trabalhos relacionados* — a fim de comparar os resultados com trabalhos relacionados. Estes cenários são descritos a seguir.

5.1 Avaliação qualitativa

Esta avaliação visa verificar a corretude e completude dos mapeamentos JSON → JSON *Schema*. Para isso, foram criados cinco documentos JSON hipotéticos (Figuras 38a, 39a, 40a, 41a e 42a), com atributos representando todos os tipos de dados JSON e JSON Estendido, ordenação das propriedades diferentes entre os documentos, bem como, diferença de tipos para uma mesma propriedade entre os documentos. Esses documentos foram armazenados em uma coleção no *MongoDB*, chamada de *allTypes*. Ao final foram analisados os mapeamentos obtidos.

As Figuras 38a e 39a apresentam documentos com a mesma estrutura e com o mesmo tipo de dados para as propriedades, sendo a única diferença os valores das propriedades. Sendo assim, a primeira operação de agregação da segunda etapa deve considerar esses documentos como possuindo o mesmo esquema bruto. As Figuras 38b e 39b apresentam, respectivamente, o esquema bruto para cada documento, obtidos na primeira etapa.

Figura 38: Documento JSON Estendido e Esquema Bruto

```
{
  "_id": ObjectId("50319491fe4dce143835c555"),
  "stringType": "some_value",
  "dateAsTime": { "$date":
     { "$numberLong": "1496795859332" }
  },
   "maxKeyType": { "$maxKey": 1 },
  "minKeyType": { "$minKey": 4 },
  "intType":1,
  "doubleType": 12343.44,
  "intType2": { "$numberInt": "1" },
  "objectType": { "property": "some_value" },
  "arrayMultiValues": [ 48, "50",
     {"property": "teste"},
     [1,"some_value"],
     true
  ],
  "arrayOfString": [ "some_value", "some_value" ],
  "arrayOfNumber": [ ],
  "arrayOfBoolean": [ true, false ],
  "arrayOfArray": [
     ["some_value", "some_value"]
  ],
  "arrayOfExtendedType": [
     Date("2014-01-31T22:26:33.000Z"),
     NumberLong("9223372036854775807")
  ]
}
```

(a) Documento JSON Estendido

```
{
  "_id": "ObjectID",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ]
}
```

(b) Esquema bruto

Fonte: Autor (2017)

Figura 39: Documento JSON Estendido e Esquema bruto

```
{
  "_id": ObjectId("50319491fe4dce143835c556"),
  "stringType": "other_value",
  "dateAsTime": { "$date":
     { "$numberLong": "1496795859444" }
  },
   "maxKeyType": { "$maxKey": 2 },
   "minKeyType": { "$minKey": 3 },
  "intType":2,
  "doubleType": 15.44,
  "intType2": { "$numberInt": "3" },
  "objectType": { "property": "other_value" },
  "arrayMultiValues": [ 30, "15",
     {"property": "teste2"},
     [2,"other_value"],
     false
  ],
  "arrayOfString": [ "other_value", "other_value" ],
  "arrayOfNumber": [ ],
  "arrayOfBoolean": [ false, true ],
  "arrayOfArray": [
     ["other_value", "other_value"]
  ],
  "arrayOfExtendedType": [
     Date("2015-01-31T22:26:33.000Z"),
     NumberLong("8223372036854775807")
  ]
}
```

(a) Documento JSON

```
{
  "_id": "ObjectID",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ]
}
```

(b) Esquema bruto

Fonte: Autor (2017)

A Figura 40a apresenta o documento mais completo do experimento, possuindo todos os tipos JSON Estendido, bem como, um *array* com itens de tipos diferentes. A

Figura 40b apresenta o esquema bruto do documento obtido na primeira etapa do processo.

Figura 40: Documento JSON Estendido e Esquema bruto

```
{
  "_id": ObjectId("50319491fe4dce143835c552"),
  "binaryType": BinData(2, "dGVzdGFuZG8="),
  "dateAsISO": Date("2014-01-31T22:26:33.000Z"),
  "timestampType": Timestamp(1421006159, 4),
  "dbRefType": DBRef("some_collection","50319491fe4dce143835c552"),
  "undefinedType":undefined,
  "longType": NumberLong("9223372036854775807"),
  "booleanType": false,
  "nullType": null,
  "regexType": /\/^S\//m,
  "stringType": "some_value",
  "dateAsTime": {
      "$date": { "$numberLong": "1496795859332" }
  },
  "objectType": { "property": "some_value" },
  "arrayMultiValues": [ 48, "50",
      {"property": "teste"},
      [1,"some_value"],
      true, BinData(2, "dGVzdGFuZG8=")
  ],
  "arrayOfObject": [ {"property": 48},
      {"property": "some_value"}
  ],
  "arrayOfString": [ "some_value", "some_value" ],
  "arrayOfNumber": [ 1, 2 ],
  "arrayOfBoolean": [ true, false ],
  "arrayOfArray": [ [1, 2],
      ["some_value", "some_value"]
  ],
  "arrayOfExtendedType": [
      Date("2014-01-31T22:26:33.000Z"),
      NumberLong("9223372036854775807")
  ],
  "javascriptType": {
      "$code": "var a = function(param){console.log(param);};"
   },
  "javascriptWithScopeType": {
      "$code": "int x = y", "$scope": { "y": 1 }
  },
   "maxKeyType": { "$maxKey": 1 },
  "minKeyType": { "$minKey": 4 },
  "intType":1,
  "doubleType": 12343.44,
  "intType2": { "$numberInt": "1" }
}
```

(a) Documento JSON Estendido

```
{
  "_id": "ObjectID",
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "dateAsTime": "Date",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean", "Binary"
  ],
  "arrayOfObject": [
    { "property": "Number" },
    { "property": "String" }
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [ "Number" ],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "Number" ],
    [ "String" ]
  ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number"
}
```

(b) Esquema bruto

Fonte: Autor (2017)

A Figura 41a apresenta um documento semelhante ao documento da Figura 40a, porém, este não possui um item do tipo *"BinData"* na propriedade *"arrayMultiValues"*. Além disso, a propriedade *"arrayOfObject"* possui apenas um item do tipo objeto, enquanto que na Figura 40a possui dois itens do tipo objeto. A Figura 41b apresenta o esquema bruto do documento obtido na primeira etapa do processo.

A Figura 42a apresenta um documento semelhante ao documento da Figura 41a, a única diferença entre os dois é a ordem das propriedades, os tipos de dados continuam os mesmos. A segunda operação de agregação da segunda etapa, na qual é realizada a ordenação das propriedades dos esquemas brutos, deve considerar esses dois documentos

Figura 41: Documento JSON Estendido e Esquema bruto

```
{
  "_id": ObjectId("50319491fe4dce143835c553"),
  "binaryType": BinData(2, "dGVzdGFuZG8="),
  "dateAsISO": Date("2014-01-31T22:26:33.000Z"),
  "timestampType": Timestamp(1421006159, 4),
  "dbRefType": DBRef("some_collection","50319491fe4dce143835c552"),
  "undefinedType":undefined,
  "longType": NumberLong("9223372036854775807"),
  "booleanType": false,
  "nullType": null,
  "regexType": /\/^S\//m,
  "stringType": "some_value",
  "dateAsTime": { "$date": {
      "$numberLong": "1496795859332" }
  },
  "objectType": { "property": "some_value" },
  "arrayMultiValues": [ 48, "50",
      {"property": "teste"},
      [1,"some_value"],
      true ],
  "arrayOfObject": [ {"property": 48} ],
  "arrayOfString": [ "some_value", "some_value" ],
  "arrayOfNumber": [ 1, 2 ],
  "arrayOfBoolean": [ true, false ],
  "arrayOfArray": [
      ["some_value", "some_value"]
  ],
  "arrayOfExtendedType": [
      Date("2014-01-31T22:26:33.000Z"),
      NumberLong("9223372036854775807")
  ],
  "javascriptType": {
      "$code": "var a = function(param){console.log(param);};"
  },
  "javascriptWithScopeType": {
      "$code": "int x = y", "$scope": { "y": 1 }
  },
  "maxKeyType": { "$maxKey": 1 },
  "minKeyType": { "$minKey": 4 },
  "intType":1,
  "doubleType": 12343.44,
  "intType2": { "$numberInt": "1" }
}
```

(a) Documento JSON Estendido

```
{
  "_id": "ObjectID",
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "dateAsTime": "Date",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number",
    "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean" ]
  ,
  "arrayOfObject": [
    { "property": "Number" }
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [ "Number" ],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number"
}
```

(b) Esquema bruto

Fonte: Autor (2017)

como possuindo o mesmo esquema bruto. A Figura 41b apresenta o esquema bruto do documento, gerado pela primeira etapa.

A primeira operação de agregação retorna quatro documentos, pois, como visto anteriormente, os documentos 38a e 39a possuem o mesmo esquema bruto. Após a ordenação das propriedades dos esquemas brutos dos quatro documentos, os documentos 42a e 41a agora possuem o mesmo esquema, sendo assim, a segunda operação de agregação retorna três documentos.

Na terceira etapa, os três documentos gerados na etapa anterior são utilizados na construção da RSUS. Como visto nas seção 4.2.3, RSUS é um documento JSON e, como tal, pode ser visualizado como uma árvore hierárquica (Figura 43). Não foi possível representar a estrutura completa devido a limites de espaço, no entanto, as propriedades

Figura 42: Documento JSON Estendido e Esquema Bruto

```
{
  "_id": ObjectId("50319491fe4dce143835c554"),
  "binaryType": BinData(2, "dGVzdGFuZG8="),
  "dateAsISO": Date("2014-01-31T22:26:33.000Z"),
  "timestampType": Timestamp(1421006159, 4),
  "dbRefType": DBRef("some_collection","50319491fe4dce143835c552"),
  "undefinedType":undefined,
  "longType": NumberLong("9223372036854775807"),
  "booleanType": false,
  "nullType": null,
  "regexType": /\/^S\//m,
  "stringType": "some_value",
  "dateAsTime": { "$date": {
      "$numberLong": "1496795859332" }
  },
   "maxKeyType": { "$maxKey": 1 },
  "minKeyType": { "$minKey": 4 },
  "intType":1,
  "doubleType": 12343.44,
  "intType2": { "$numberInt": "1" },
  "objectType": { "property": "some_value" },
  "arrayMultiValues": [  48, "50",
      {"property": "teste"},
      [1,"some_value"],
      true
  ],
  "arrayOfObject": [ {"property": 48} ],
  "arrayOfString": [ "some_value", "some_value" ],
  "arrayOfNumber": [ 1, 2 ],
  "arrayOfBoolean": [ true, false ],
  "arrayOfArray": [
      ["some_value", "some_value"]
  ],
  "arrayOfExtendedType": [
      Date("2014-01-31T22:26:33.000Z"),
      NumberLong("9223372036854775807")
  ],
  "javascriptType": {
      "$code": "var a = function(param){console.log(param);};"
  },
  "javascriptWithScopeType": {
      "$code": "int x = y", "$scope": { "y": 1 }
  }
}
```

(a) Documento JSON Estendido

```
{
  "_id": "ObjectID",
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfObject": [
    { "property": "Number" }
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [ "Number" ],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [
    [ "String" ]
  ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code"
}
```

(b) Esquema bruto

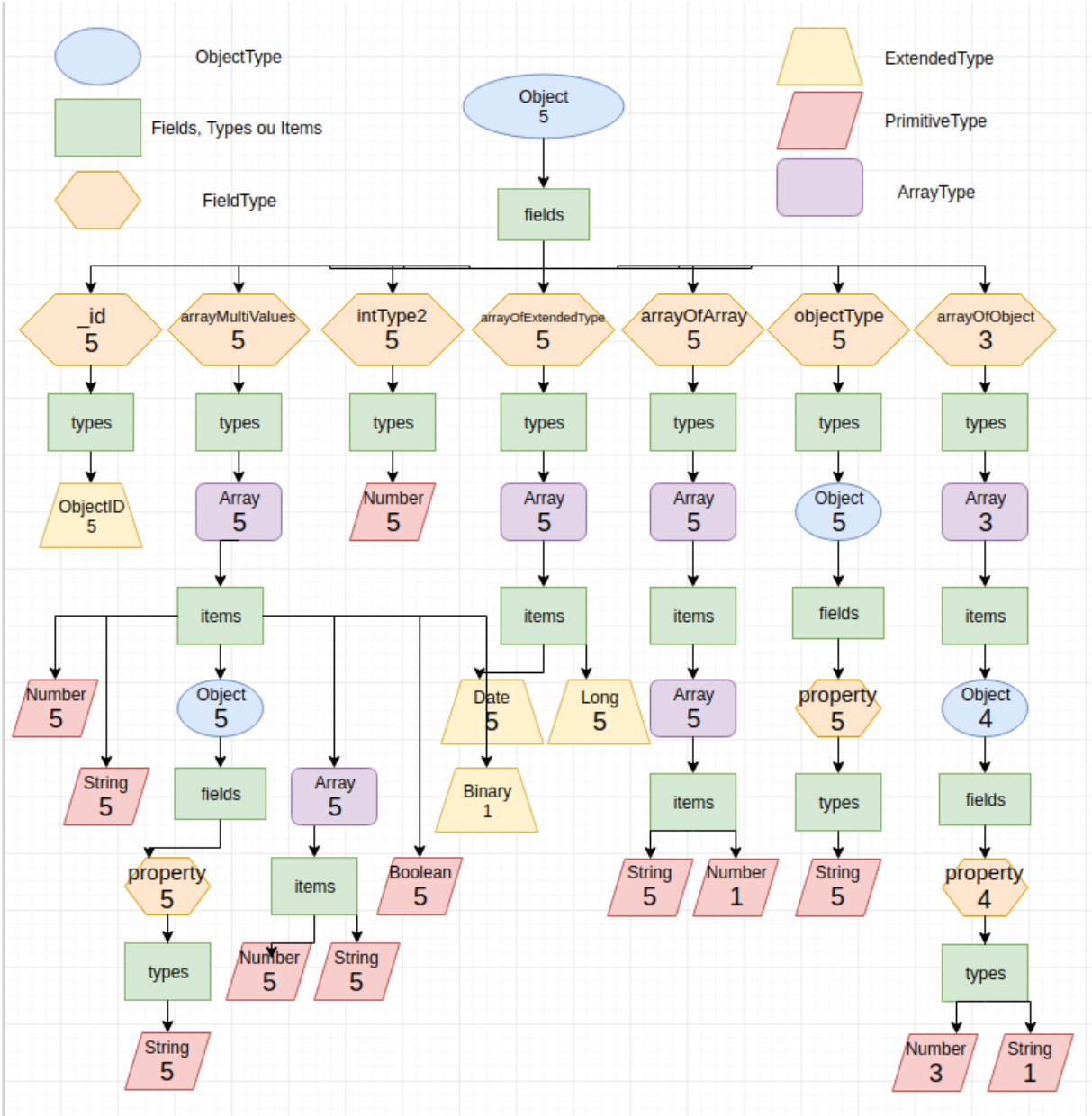
Fonte: Autor (2017)

mais complexas são apresentadas. Os números presentes nas propriedades são utilizados para definir se a propriedade é obrigatória ou opcional.

Por fim, na quarta etapa há o mapeamento RSUS → JSON *Schema*. O resultado pode ser visto nas Figuras 44, 45, 46, 47 e 48. Os tipos do JSON Estendido são definidos na seção de definições do esquema e são referenciados nas propriedades. A Figura 44 monstra tais definições. As Figuras 45, 46, 47 e 48 apresentam a definição de esquema para as propriedades dos cinco documentos analisados.

Quanto a qualidade do esquema esquema gerado, obteve-se uma precisão de 100%, visto que foram identificados todos os tipos de dados esperados nos documentos. Estes tipos de dados incluem atributos obrigatórios, tipos JSON Estendido, quantidade de

Figura 43: Visualização da RSUS como uma árvore hierárquica



Fonte: Autor (2017)

Figura 44: JSON *Schema* - Definições

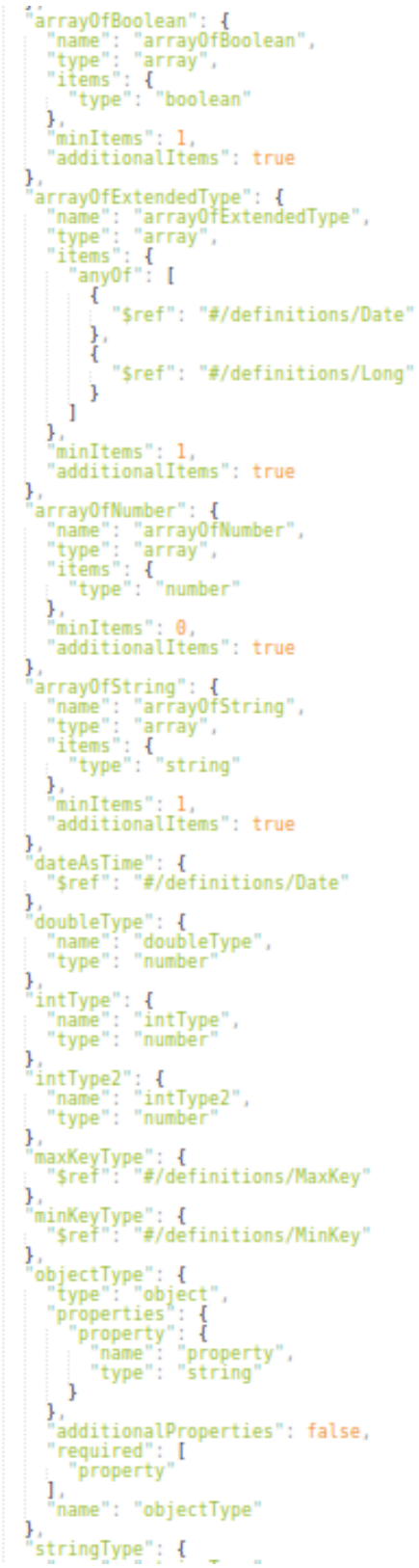
```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "Binary": {
      "type": "object",
      "properties": { "$binary": { "type": "string" }, "$type": { "type": "string" } },
      "required": [ "$binary", "$type" ]
    },
    "Undefined": {
      "type": "object",
      "properties": { "$undefined": { "type": "boolean" } },
      "required": [ "$undefined" ]
    },
    "ObjectID": {
      "type": "object",
      "properties": { "$oid": { "type": "string" } },
      "required": [ "$oid" ]
    },
    "Date": {
      "type": "object",
      "properties": { "$date": { "type": "string" } },
      "required": [ "$date" ]
    },
    "RegExp": {
      "type": "object",
      "properties": { "$options": { "type": "string" }, "$regex": { "$type": "string" } },
      "required": [ "$options", "$regex" ]
    },
    "DBRef": {
      "type": "object",
      "properties": { "$id": { "type": "string" }, "$ref": { "type": "string" } },
      "required": [ "$id", "$ref" ]
    },
    "Code": {
      "type": "object",
      "properties": { "$code": { "type": "string" }, "$scope": { "type": "object" } },
      "required": [ "$code" ]
    },
    "Timestamp": {
      "type": "object",
      "properties": {
        "$timestamp": {
          "type": "object",
          "properties": { "i": { "type": "integer" }, "t": { "type": "integer" } },
          "required": [ "i", "t" ]
        }
      },
      "required": [ "$timestamp" ]
    },
    "Long": {
      "type": "object",
      "properties": { "$numberLong": { "type": "string" } },
      "required": [ "$numberLong" ]
    },
    "MinKey": {
      "type": "object",
      "properties": { "$minKey": { "type": "integer" } },
      "required": [ "$minKey" ]
    },
    "MaxKey": {
      "type": "object",
      "properties": { "$maxKey": { "type": "integer" } },
      "required": [ "$maxKey" ]
    }
  },
  "properties": {•••
  },
  "additionalProperties": false,
  "required": [•••
  ]
}
```

Fonte: Autor (2017)

Figura 45: JSON *Schema* - Propriedades 1/4

Fonte: Autor (2017)

Figura 46: JSON *Schema* - Propriedades 2/4



```
{
"arrayOfBoolean": {
  "name": "arrayOfBoolean",
  "type": "array",
  "items": {
    "type": "boolean"
  },
  "minItems": 1,
  "additionalItems": true
},
"arrayOfExtendedType": {
  "name": "arrayOfExtendedType",
  "type": "array",
  "items": {
    "anyOf": [
      {
        "$ref": "#/definitions/Date"
      },
      {
        "$ref": "#/definitions/Long"
      }
    ]
  },
  "minItems": 1,
  "additionalItems": true
},
"arrayOfNumber": {
  "name": "arrayOfNumber",
  "type": "array",
  "items": {
    "type": "number"
  },
  "minItems": 0,
  "additionalItems": true
},
"arrayOfString": {
  "name": "arrayOfString",
  "type": "array",
  "items": {
    "type": "string"
  },
  "minItems": 1,
  "additionalItems": true
},
"dateAsTime": {
  "$ref": "#/definitions/Date"
},
"doubleType": {
  "name": "doubleType",
  "type": "number"
},
"intType": {
  "name": "intType",
  "type": "number"
},
"intType2": {
  "name": "intType2",
  "type": "number"
},
"maxKeyType": {
  "$ref": "#/definitions/MaxKey"
},
"minKeyType": {
  "$ref": "#/definitions/MinKey"
},
"objectType": {
  "type": "object",
  "properties": {
    "property": {
      "name": "property",
      "type": "string"
    }
  },
  "additionalProperties": false,
  "required": [
    "property"
  ],
  "name": "objectType"
},
"stringType": {
```

Fonte: Autor (2017)

Figura 47: JSON *Schema* - Propriedades 3/4

```
},
"stringType": {
  "name": "stringType",
  "type": "string"
},
"arrayOfObject": {
  "name": "arrayOfObject",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "property": {
        "name": "property",
        "anyOf": [
          {
            "type": "number"
          },
          {
            "type": "string"
          }
        ]
      }
    },
    "additionalProperties": false,
    "required": [
      "property"
    ]
  },
  "minItems": 1,
  "additionalItems": true
},
"binaryType": {
  "$ref": "#/definitions/Binary"
},
"booleanType": {
  "name": "booleanType",
  "type": "boolean"
},
"dateAsISO": {
  "$ref": "#/definitions/Date"
},
"dbRefType": {
  "$ref": "#/definitions/DBRef"
},
"javascriptType": {
  "$ref": "#/definitions/Code"
},
"javascriptWithScopeType": {
  "$ref": "#/definitions/Code"
},
"longType": {
  "$ref": "#/definitions/Long"
},
"nullType": {
  "name": "nullType",
  "type": "null"
},
"regexType": {
  "$ref": "#/definitions/RegExp"
},
"timestampType": {
  "$ref": "#/definitions/Timestamp"
},
"undefinedType": {
  "$ref": "#/definitions/Undefined"
}
},
"additionalProperties": false,
```

Fonte: Autor (2017)

Figura 48: JSON *Schema* - Propriedades 4/4

```
  },
  "additionalProperties": false,
  "required": [
    "_id",
    "arrayMultiValues",
    "arrayOfArray",
    "arrayOfBoolean",
    "arrayOfExtendedType",
    "arrayOfNumber",
    "arrayOfString",
    "dateAsTime",
    "doubleType",
    "intType",
    "intType2",
    "maxKeyType",
    "minKeyType",
    "objectType",
    "stringType"
  ]
}
```

Fonte: Autor (2017)

itens mínimos em um *array* e tipos de união para quando uma propriedade possui mais de um tipo de dado. Vale destacar o significado dos atributos *"additionalProperties"* e *"additionalItems"*: o primeiro foi definido com o valor *false*, pois o processo é aplicado em todos os documentos da coleção, ou seja, todos os atributos existentes nos documentos do *dataset* já estão definidos no esquema; o segundo foi definido como *true*, pois um *array* pode ter mais de um item.

5.2 Análise do tempo de processamento

Nesta seção são apresentados os resultados da análise de tempo de processamento da extração de esquemas para coleções massivas de dados (*massive datasets*). Os experimentos foram executados em uma instância *Amazon* EC2 *t2.micro* (Intel® Xeon® E5-2676 v3 @ 2.40GHz e 1GB de memória RAM).

Os *datasets* utilizados no experimento foram obtidos de um trabalho existente (Çelikten, Falher e Mathioudakis (2017)) e correspondem a dados rastreados de *"tweets"*, *"checkins"* e locais (*"venues"*) do *Foursquare*. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

Na maioria dos cenários de manipulação de grande quantidade de dados, processos são executados em lote, portanto, não é exigido tempos de execução que permitam interações de usuários em tempo real. No entanto, o tempo de execução é ainda um critério

crítico para a viabilidade da extração de esquema. Sendo assim, foram realizados diversos experimentos e usadas diferentes técnicas de programação, com o objetivo de se otimizar o tempo total de extração durante o desenvolvimento da ferramenta.

Os resultados experimentais da Tabela 2 mostram o tempo de processamento gasto em média no processo de extração de esquema para os três *datasets*. Vale destacar a importância da etapa de ordenação das propriedades do esquema bruto, como observado para o *dataset "venues"*, em que a quantidade real de esquemas brutos distintos era de 117, ao invés de 257 obtidos antes da ordenação das propriedades. Vale ressaltar, também, o tempo de processamento gasto para leitura dos documentos e geração dos esquemas brutos, chegando a 99% do tempo total do processo, visto que é realizado a leitura de todos os documentos de uma coleção. Várias técnicas foram utilizadas e testadas nesta etapa, como o uso de *streams* e processamento assíncrono, visando a otimização do tempo total. Esse tempo pode ser melhorado, em trabalhos futuros, melhorando a configuração da máquina de testes e usando *clusters* para processamento paralelo.

Tabela 2: Resultados para os *datasets venues*, *tweets* e *checkins*

Coleção	Número de documentos	Esquema bruto	Esquema bruto com estrutura ordenada	Tempo obtenção dos esquemas brutos (TB)	TB/TT	Tempo total (TT)
venues	2.096.353	257	117	7,47 min	99,33%	7,52 min
checkings	11.577.228	2	2	35,27 min	99,29%	35,52 min
tweets	16.972.245	23	16	53,11 min	99,38%	53,44 min

Fonte: Autor (2017)

5.3 Comparação com trabalhos relacionados

Nesta seção são apresentados os resultados da análise do processo de extração de esquemas para *datasets* utilizados em trabalhos relacionados. Os experimentos foram executados em um *notebook* ASUS K45VM (*Intel*® *Core*™ i7 3610QM @ 2.30 GHz e 8GB de memória RAM).

Os *datasets* (*drugs*, *companies* e *movies*) utilizados no primeiro experimento (tabela 3) foram obtidos do trabalho de Wang et al. (2015) (ver seção 3.3) e correspondem

a dados retirados da *DBPedia*[1]. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

Estes *datasets* possuem vários esquemas brutos distintos, como pode ser visualizado nas colunas "Esquemas brutos" e "Esquemas brutos com estrutura ordenada" da Tabela 3, chegando o número a ser muito próximos da quantidade de documentos do *dataset* (coluna "Número de documentos"). De acordo com Wang et al. (2015), isso se deve à natureza dos documentos do *DBPedia*, cuja diferença entre documentos, geralmente é de alguns poucos atributos.

Quanto a quantidade de esquemas distintos identificados, ambas propostas chegaram a mesma quantidade para o *dataset drugs*, enquanto que para os *datasets companies* e *movies*, o presente trabalho identificou uma quantidade maior. Essa diferença ocorreu, pois ao contrário da proposta de Wang et al. (2015), a presente proposta leva em consideração o nome e tipo de cada propriedade de um documento para montar seu respectivo esquema bruto (ver seção 4.2.1), e não apenas os nomes das propriedades, como é realizado na criação do esquema de registro (ver seção 3.3), utilizado no processo proposto por Wang et al. (2015). O tipo do dado também é importante para permitir que se faça consultas mais exatas em relação a filtros sobre determinados atributos. Por exemplo, se o tipo de dado de um atributo é numérico, o usuário poderá definir um filtro envolvendo uma constante numérica.

Tabela 3: Resultados para os *datasets drugs*, *companies* e *movies*

Datasets DBPedia		JSONSchema Discovery				WANG et. al. (2015)
Coleção	Número de documentos	Esquemas brutos	Esquemas brutos com estrutura ordenada	Tempo de obtenção dos esquemas brutos	Tempo total	Esquemas
drugs	3662	2818	2818	1,95 s	5,1 s	2818
companies	24367	21312	21312	11,43 s	30 s	21302
movies	30330	25140	25140	14,92 s	39 s	25137

Fonte: Autor (2017)

O *dataset* (*pullrequest*) utilizado no segundo experimento (Tabela 4) foi obtido de Baazizi et al. (2017) e corresponde a dados retirados do *GitHub*[2]. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

[1] http://wiki.dbpedia.org/
[2] https://github.com/

Tabela 4: Resultados para o *dataset pullrequests*

.

Dataset GitHub		JSONSchema Discovery				Baazizi et. al. (2017)
Coleção	Número de documentos	Esquemas brutos	Esquemas brutos com estrutura ordenada	Tempo de obtenção dos esquemas brutos	Tempo total	Esquemas
pullrequests	1000	29	29	1,12 s	1,61 s	29
pullrequests	10000	66	66	7,38 s	8,4 s	66
pullrequests	100000	261	261	1,21 min	1,24 min	261

Fonte: Autor (2017)

O *dataset* do *GitHub* corresponde a metadados gerados com *pull requests* emitidos por usuários. Três testes foram realizados, selecionando mil, dez mil e cem mil documentos, para se comparar a quantidade de esquemas identificados em cada teste. Em ambas as propostas foram identificados a mesma quantidade de esquemas.

6 Conclusão

Este trabalho de conclusão de curso teve como objetivo principal o desenvolvimento de uma ferramenta *web* para extração de esquemas de documentos JSON armazenados em um BD orientado a documentos.

A principal motivação para este trabalho é facilitar tarefas como a manipulação e a integração de dados, pois com um esquema explicito definido tem-se o conhecimento geral das propriedades obrigatórias e opcionais existentes em uma coleção de documentos, bem como, o tipo de dado permitido em cada propriedade. A definição de um esquema também pode ser utilizado na validação de dados, como visto na seção 2.5 e facilita também no desenvolvimento de *software*, visto que restrições de esquema não precisam ser especificadas dentro da lógica da aplicação.

Para atingir este objetivo, foi realizado um levantamento do estado da arte sobre *"extração de esquemas de BDs NoSQL"*, um estudo teórico sobre as tecnologias envolvidas no processo de extração de esquemas de BDs NoSQL, especificamente BDs de documento e documentos no formato JSON, a fim de levantar os requisitos e potenciais diferenciais da ferramenta, batizada de *JSONSchema Discovery*.

A *JSONSchema Discovery* destaca-se das demais propostas pelas seguintes razões: é uma ferramenta *web*, estando, deste modo, disponível a qualquer dispositivo com acesso a um navegador; utiliza uma especificação padrão para definição de esquemas de documentos JSON (JSON *Schema* (ver seção 2.5)); armazena os esquemas gerados; mantém o histórico dos esquemas gerados; permite a visualização do JSON *Schema* gerado; oferece suporte à JSON estendido (ver seção 2.4.1); e apresenta uma interface gráfica simples e intuitiva, além de ser totalmente gratuita.

Os objetivos específicos (seção 1.2.2) podem ser considerados como atingidos. O objetivo especifico 1, que tratou a extração dos esquemas brutos dos documentos, pode ser visto em detalhes na seção 4.2.1. O objetivo 2, que tratou o mapeamento dos esquemas brutos para um único esquema no padrão JSON *Schema*, pode ser visto em detalhes nas seções 4.2.2, 4.2.3 e 4.2.4. Por fim, o objetivo especifico 3, que trata da avaliação da ferramenta proposta, pode ser visto no capítulo 5.

Assim, pode-se concluir que o objetivo geral (seção 4.1) do projeto também foi atingido, visto que a ferramenta foi projetada com técnicas modernas de desenvolvimento *web*, que possibilitaram o cumprimento dos requisitos funcionais e não funcionais apresentados na seção 4.1 e no Apêndice A e demonstrados na seção 4.3.3, resultando em uma ferramenta leve, escalável e fácil de usar. Como essa ferramenta foi concebida para extrair esquemas de documentos JSON armazenados no SGBD *MongoDB*, já se assume que os documentos estão bem formados. Entretanto, para o caso dessa ferramenta ser estendida para extrair esquemas de qualquer fonte de dados JSON, sua limitação é não ser capaz de extrair esquemas de documentos JSON mal formados. Pretende-se tratar essa limitação em trabalhos futuros, conforme descrito a seguir.

Os temas e tecnologias aqui abordados acrescentaram grande conhecimento para seu autor, pois uma vasta pesquisa em BDs NoSQL, JSON, JSON *Schema* e *Software-as-a-Service* foi necessária para o desenvolvimento da ferramenta. Um conhecimento aprofundado em *MongoDB* foi adquirido neste trabalho também. Tal aquisição foi necessária pois as etapas de extração, requisitaram funcionalidades avançadas deste banco, como: *stream* de dados, utilizado na etapa 1, e operações de agregação, utilizadas na etapa 2.

Algumas funcionalidades podem ser adicionadas, a fim de agregar maior maturidade à ferramenta, possibilitando assim sua melhoria contínua. Dessa forma, apresentam-se as seguintes possibilidades de trabalhos futuros:

- Melhorar a *performance* da primeira etapa de extração (seção 4.2.1);
- Geração de esquemas para outros SGBDs NoSQL;
- Permitir a opção de *upload* de um ou mais documentos JSON para extração, considerando a existência de um *parser* que verifica se os documentos estão bem formados;
- Identificar e tratar relacionamentos por referência, como realizado na proposta de Ruiz, Morales e Molina (2015);
- Internacionalização do *software*, gerando versões em língua inglesa e espanhola.

Os *links* para a ferramenta podem ser encontrados no repositório *GitHub*[1] da aplicação, no qual é possível encontrar também todo o código, fazer sugestões e até colaborar com novas implementações.

[1] https://github.com/feekosta/JSONSchemaDiscovery

Referências

BAAZIZI, M.-a. et al. Schema Inference for Massive JSON Datasets. *Edbt*, p. 222–233, jan 2017. Disponível em: <https://openproceedings.org/2017/conf/edbt/paper-62.pdf>. Citado na página 99.

Big Data is the new black. *What is Big Data*. 2016. Disponível em: <http://bigdata-.black/featured/what-is-big-data/>. Acesso em: 16 jun. 2017. Citado na página 34.

BOICEA, A.; RADULESCU, F.; AGAPIN, L. I. MongoDB vs Oracle - Database comparison. *Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012*, p. 330–335, 2012. Citado na página 44.

BREWER, E. A. Towards robust distributed systems (abstract). *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*, v. 19, p. 7, 2000. ISSN 01635700. Disponível em: <http://portal-.acm.org/citation.cfm?doid=343477.343502>. Citado 3 vezes nas páginas 36, 37 e 38.

BREWER, E. A. CAP twelve years later: How the "rules"have changed. *Computer*, v. 45, n. 2, p. 23–29, 2012. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org-/document/6133253/>. Citado 2 vezes nas páginas 37 e 38.

bson-spec-org. *BSON*. 2017. Disponível em: <http://bsonspec.org/>. Acesso em: 26 out. 2017. Citado na página 46.

CATTELL, R. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, v. 39, n. 4, p. 12, 2011. ISSN 01635808. Disponível em: <http://portal.acm.org/citation-.cfm?doid=1978915.1978919>. Citado na página 27.

CROCKFORD, D. *JavaScript: The Good Parts*. [S.l.]: O'Reilly Media, Inc., 2008. ISBN 0596517742. Citado na página 78.

ECMA-404. The JavaScript Object Notation (JSON) Data Interchange Format. *ECMA International*, v. 1st Editio, n. October, p. 8, 2014. ISSN 2070-1721. Disponível em: <https://www.rfc-editor.org/info/rfc7159>. Citado 3 vezes nas páginas 44, 45 e 46.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado), 2000. AAI9980887. Citado na página 80.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420. Citado na página 81.

GYORODI, C. et al. A comparative study: MongoDB vs. MySQL. *2015 13th International Conference on Engineering of Modern Electric Systems, EMES 2015*, p. 0–5, 2015. Citado na página 44.

HAN, J. et al. Survey on NoSQL database. *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, p. 363–366, 2011. ISSN 978-1-4577-0207-5. Citado 5 vezes nas páginas 27, 28, 33, 42 e 43.

HASHEM, H.; RANC, D. Evaluating NoSQL document oriented data model. *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*, p. 51–56, 2016. Disponível em: <http://-/ieeexplore.ieee.org/document/7592700/>. Citado 4 vezes nas páginas 28, 39, 40 e 43.

IETF. *RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format* . 2014. Disponível em: <https://tools.ietf.org/html/rfc7159>. Acesso em: 29 jun. 2017. Citado na página 30.

IMHOF, R.; FROZZA, A. A.; MELLO, R. d. S. Um Survey sobre Extração de esquemas de documentos json. *Escola Regional de Banco de Dados - ERBD 2017*, p. 26–35, 2017. Citado na página 63.

INDRAWAN-SANTIAGO, M. Database research: Are we at a crossroad? Reflection on NoSQL. *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012*, p. 45–51, 2012. Citado 3 vezes nas páginas 28, 42 e 43.

IZQUIERDO, J. L. C.; CABOT, J. Discovering implicit schemas in json data. In: *Proceedings of the 13th International Conference on Web Engineering*. Berlin, Heidelberg: Springer-Verlag, 2013. (ICWE'13), p. 68–83. ISBN 978-3-642-39199-6. Disponível em: <http://dx.doi.org/10.1007/978-3-642-39200-9_8>. Citado 6 vezes nas páginas 16, 51, 59, 60, 61 e 64.

JMTAURO, C.; S, A.; A.B, S. Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. *International Journal of Computer Applications*, v. 48, n. 20, p. 1–4, 2012. ISSN 09758887. Disponível em: <http://research-.ijcaonline.org/volume48/number20/pxc3880336.pdf>. Citado 2 vezes nas páginas 41 e 43.

JSON Schema. *Specification Links*. 2017. Disponível em: <http://json-schema.org-/specification-links.html>. Acesso em: 29 jun. 2017. Citado na página 30.

KAISLER, S. et al. Big Data: Issues and Challenges Moving Forward. *2013 46th Hawaii International Conference on System Sciences*, p. 995–1004, 2013. ISSN 1530-1605. Disponível em: <http://ieeexplore.ieee.org/document/6479953/>. Citado na página 35.

KATAL, A.; WAZID, M.; GOUDAR, R. H. Big data: Issues, challenges, tools and Good practices. *2013 6th International Conference on Contemporary Computing, IC3 2013*, p. 404–409, 2013. Citado 2 vezes nas páginas 34 e 35.

KLETTKE, M.; STÖRL, U.; SCHERZINGER, S. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In: *Btw*. [S.l.]: GI, 2015. (LNI, v. 241), p. 425–444. ISBN 9783885796350. ISSN 16175468. Citado 7 vezes nas páginas 15, 29, 51, 52, 53, 64 e 65.

LARMAN, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN 0131489062. Citado na página 67.

PEZOA, F. et al. Foundations of JSON Schema. In: *Proceedings of the 25th International Conference on World Wide Web - WWW '16*. International World Wide Web Conferences Steering Committee, 2016. (WWW '16), p. 263–273. ISBN 9781450341431. Disponível em: <http://dl.acm.org/citation.cfm?doid=2872427.2883029>. Citado 3 vezes nas páginas 46, 47 e 48.

RUIZ, D. S.; MORALES, S. F.; MOLINA, J. G. Inferring Versioned Schemas from NoSQL Databases and its Applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 9381, n. October, p. 467–480, 2015. ISSN 16113349. Disponível em: <http://link.springer.com/10.1007/978-3-319-25264-3_35>. Citado 11 vezes nas páginas 15, 27, 28, 51, 54, 55, 56, 57, 64, 69 e 102.

SADALAGE, P. J.; FOWLER, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. [S.l.: s.n.], 2012. 192 p. ISSN 1098-. ISBN 9780321826626. Citado 11 vezes nas páginas 27, 28, 29, 33, 35, 36, 38, 39, 40, 41 e 43.

SAGIROGLU, S.; SINANC, D. Big data: A review. *2013 International Conference on Collaboration Technologies and Systems (CTS)*, p. 42–47, 2013. ISSN 9781467364034. Disponível em: <http://ieeexplore.ieee.org/document/6567202/>. Citado 2 vezes nas páginas 34 e 35.

SILVA, R. P. e. *Como modelar com UML 2*. [S.l.]: Visual Books, 2009. ISBN 8575022431. Citado na página 68.

TANENBAUM, A. S.; WETHERALL, D. J. *Computer Networks*. 5th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0132126958, 9780132126953. Citado 2 vezes nas páginas 80 e 81.

TUDORICA, B. G.; BUCUR, C. A comparison between several NoSQL databases with comments and notes. *Proceedings - RoEduNet IEEE International Conference*, 2011. ISSN 20681038. Citado 2 vezes nas páginas 27 e 38.

WANG, L. et al. Schema management for document stores. *Proceedings of the VLDB Endowment*, v. 8, n. 9, p. 922–933, 2015. ISSN 21508097. Disponível em: <http://dl.acm.org/citation.cfm?doid=2777598.2777601>. Citado 8 vezes nas páginas 15, 51, 57, 58, 59, 64, 98 e 99.

Wikimedia. *Software framework*. 2017. Disponível em: <https://en.wikipedia-.org/wiki/Software_framework>. Acesso em: 29 jun. 2017. Citado na página 57.

WISCHENBART, M. et al. User profile integration made easy: Model-driven extraction and transformation of social network schemas. In: *Proceedings of the 21st International Conference on World Wide Web*. New York, NY, USA: ACM, 2012. (WWW '12 Companion), p. 939–948. ISBN 978-1-4503-1230-1. Disponível em: <http://doi.acm.org/10.1145/2187980.2188227>. Citado 6 vezes nas páginas 16, 29, 51, 62, 63 e 64.

ÇELIKTEN, E.; FALHER, G. L.; MATHIOUDAKIS, M. Modeling urban behavior by mining geotagged social data. *IEEE Transactions on Big Data*, v. 3, n. 2, p. 220–233, June 2017. Citado na página 97.

Apêndices

APÊNDICE A – Análise de Requisitos

Especificação de requisitos

Projeto – JSONSchema Discovery

1 Especificações de Requisitos de Software

Versão 1.0 04/08/2017

Versão	Autor	Data	Ação
1.0	Felipe de Souza da Costa	04/08/2017	Estabelecimento dos requisitos

2 Conteúdo

1. Introdução

1.1 Objetivo do Desenvolvimento

Desenvolvimento de uma aplicação que possibilite a extração do esquema de uma coleção de documentos no formato JSON, armazenados em um banco de dados NoSQL de documento.

1.2 Descrição da aplicação:

JSONSchema Discovery é uma aplicação web para extração de esquemas – no formato padrão JSON Schema - de uma coleção de documentos armazenados em um banco de dados orientado a documentos MongoDB. A aplicação mantém o histórico de esquemas extraídos para cada usuário e o mesmo pode visualizar o esquema extraído, bem como baixar o resultado.

2. Visão Geral:

2.1 Arquitetura do programa:

- Software-as-a-Service

2.2 Premissas de desenvolvimento:

- A ferramenta deverá apresentar uma interface gráfica, onde o usuário deve interagir com o uso do *mouse* e, ou do teclado;
- A ferramenta deverá ser desenvolvida com a linguagem de programação Javascript.
- A ferramenta deverá persistir os resultados, bem como as informações do usuário.

3. Requisitos de Software

3.1. Requisitos funcionais

Requisito funcional 01 – Criar conta: a ferramenta deve apresentar a opção "*Criar conta*" para que o usuário efetue o registro para ter acesso à ferramenta.

Requisito funcional 02 – Entrar: a ferramenta deve apresentar a opção "*Entrar*" para estabelecer uma sessão com o servidor, de modo que seja possível a interação com a ferramenta. A ferramenta notificará se a sessão foi estabelecida ou não;

Requisito funcional 03 – Sair: a ferramenta deve apresentar a opção "*Sair*" permitindo encerrar a sessão com o servidor. Ao selecionar esta opção, a sessão com o servidor deverá ser encerrada;

Requisito funcional 04 – Extrair esquema: a ferramenta deve apresentar a opção "*Extrair esquema*" permitindo que o usuário informe os dados de conexão com o banco de dados, bem como a coleção da qual se deseja obter o esquema. Considera-se a seguinte restrição:

- Os dados de conexão com o banco de dados devem ser válidos;
- A coleção informada deve existir no banco de dados informado, bem como deve conter pelo menos um documento;

Requisito funcional 05 – Deletar resultado: a ferramenta deve apresentar a opção "Deletar *resultado*" permitindo que o usuário delete um resultado, e este não seja mais visualizado na ferramenta;

Requisito funcional 06 – Visualizar resultado: a ferramenta deve apresentar a opção "*Visualizar resultado*" permitindo que o usuário visualize o JSON Schema gerado para a coleção de documentos informado;

Requisito funcional 07 – Baixar resultado: a ferramenta deve apresentar a opção "*Baixar resultado*" permitindo que o usuário baixe o JSON Schema gerado. Considera-se a seguinte restrição:

- O nome do arquivo deve ser o nome da coleção;
- O formato do arquivo deve ser .json;

Requisito funcional 08 – Listar os resultados: a ferramenta deve apresentar a lista de esquemas gerados, informando o status de cada resultado (erro, sucesso, em andamento). Permitindo assim que o usuário visualize o histórico de esquemas extraídos;

Requisito funcional 09 – Notificação de resultado: a ferramenta deve notificar o resultado da extração de esquema para o usuário (sucesso ou erro);

3.2. Requisitos não funcionais

Requisito não funcional 01 – Arquitetura: A arquitetura da ferramenta deve ser desenvolvida no formato *client-server*;

Requisito não funcional 02 – Linguagem de programação: A ferramenta deve ser desenvolvida em Typescript;

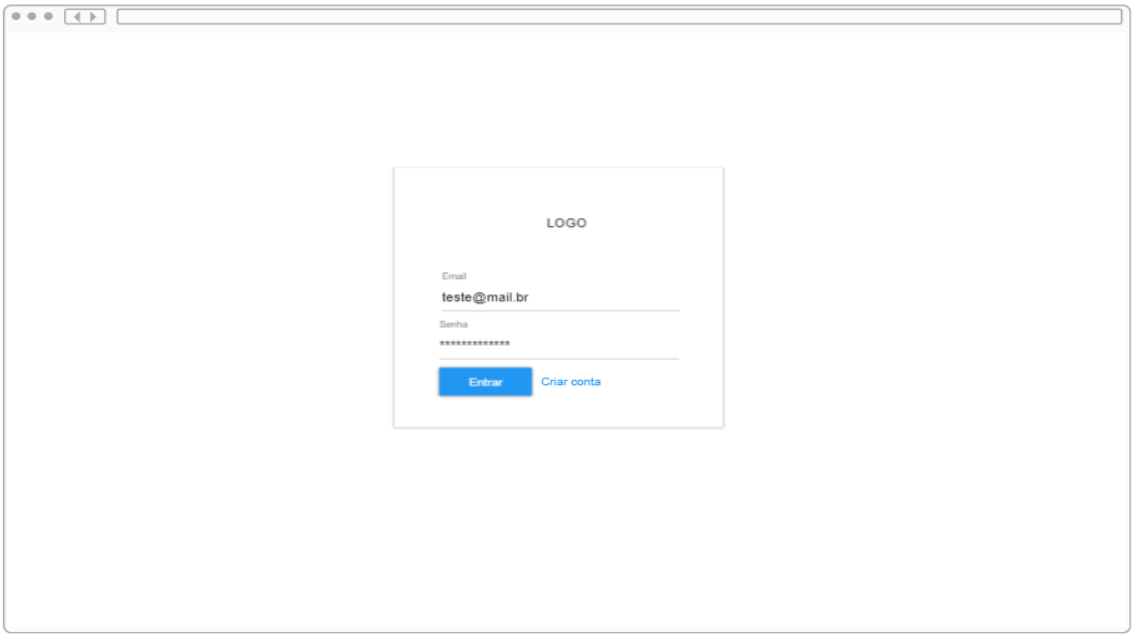
Requisito não funcional 03 – Front-end: O front-end deve ser desenvolvido com o framework Angular, bem como utilizar os componentes do Angular Material;

Requisito não funcional 04 – Back-end: O back-end deve ser desenvolvido com os frameworks NodeJS e ExpressJS;

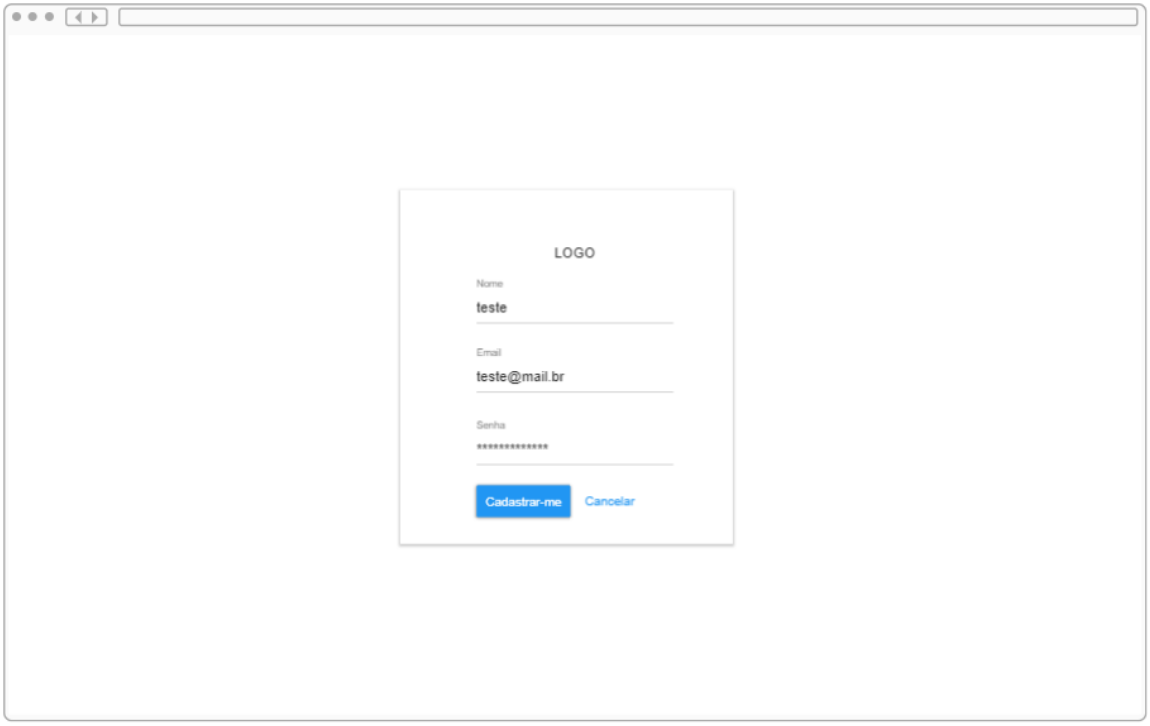
Requisito não funcional 05 – Banco de dados: O banco de dados a ser utilizado na ferramenta deve ser o MongoDB.

4. Esboço da Interface Gráfica

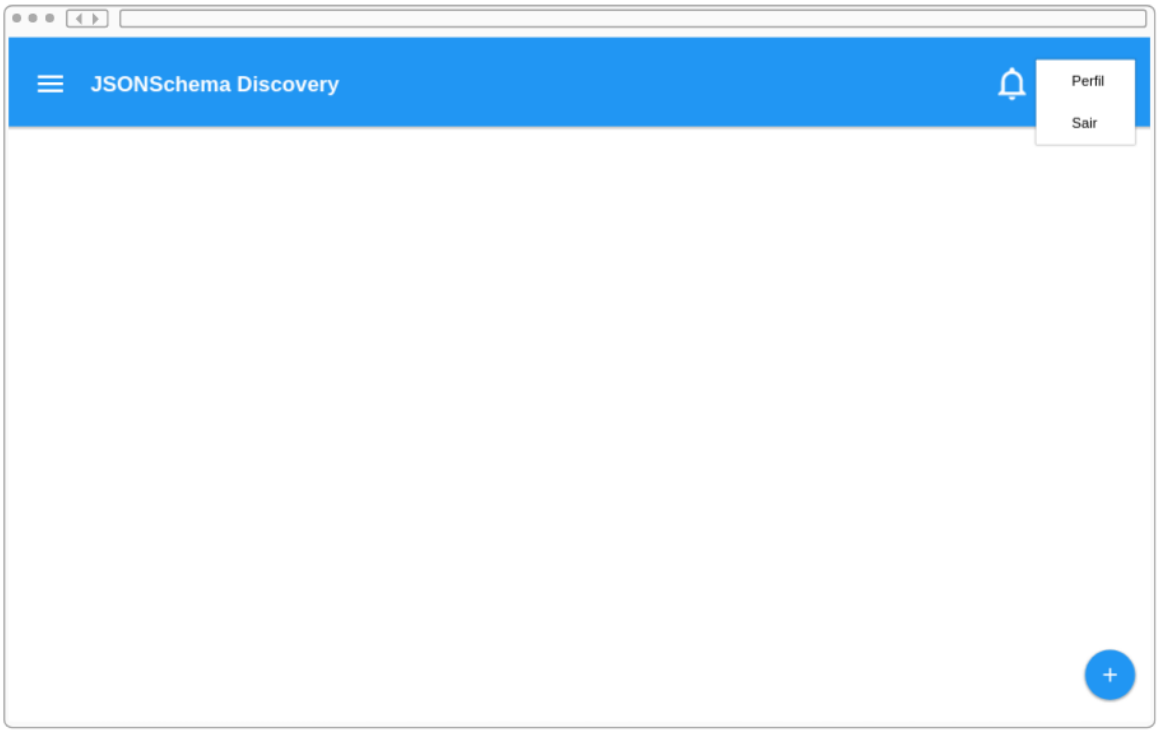
4.1 Efetuar login (RF02)



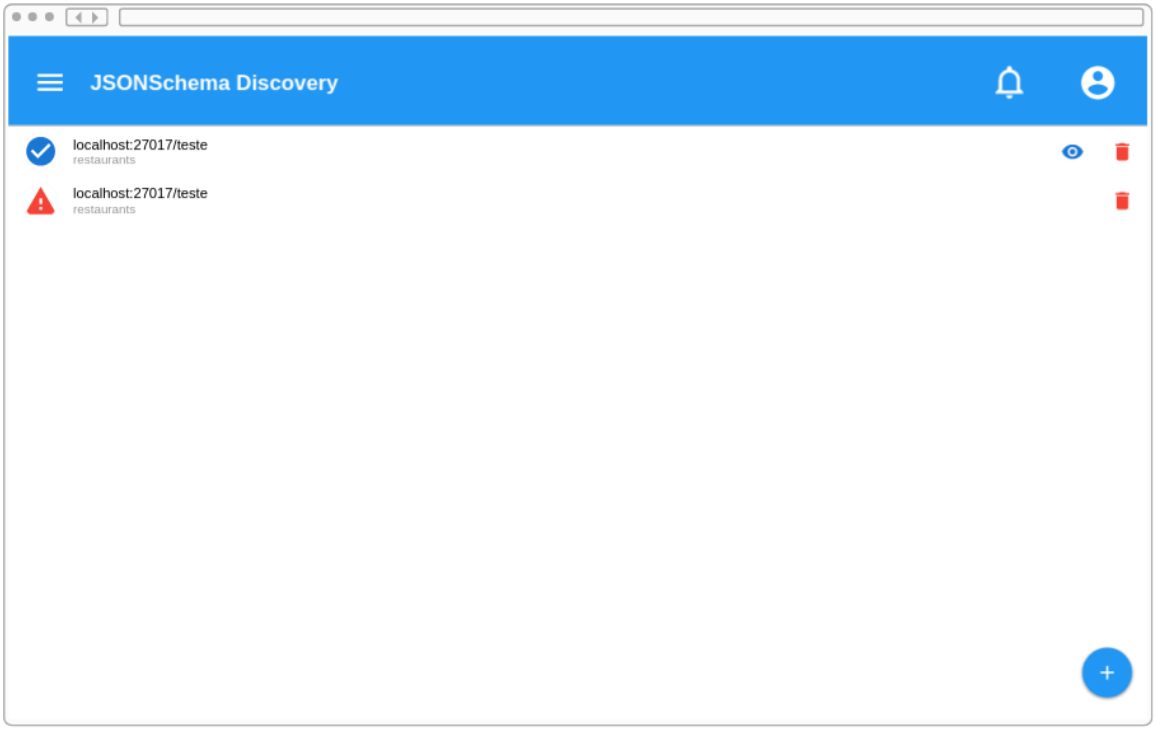
4.2 Criar conta (RF01)



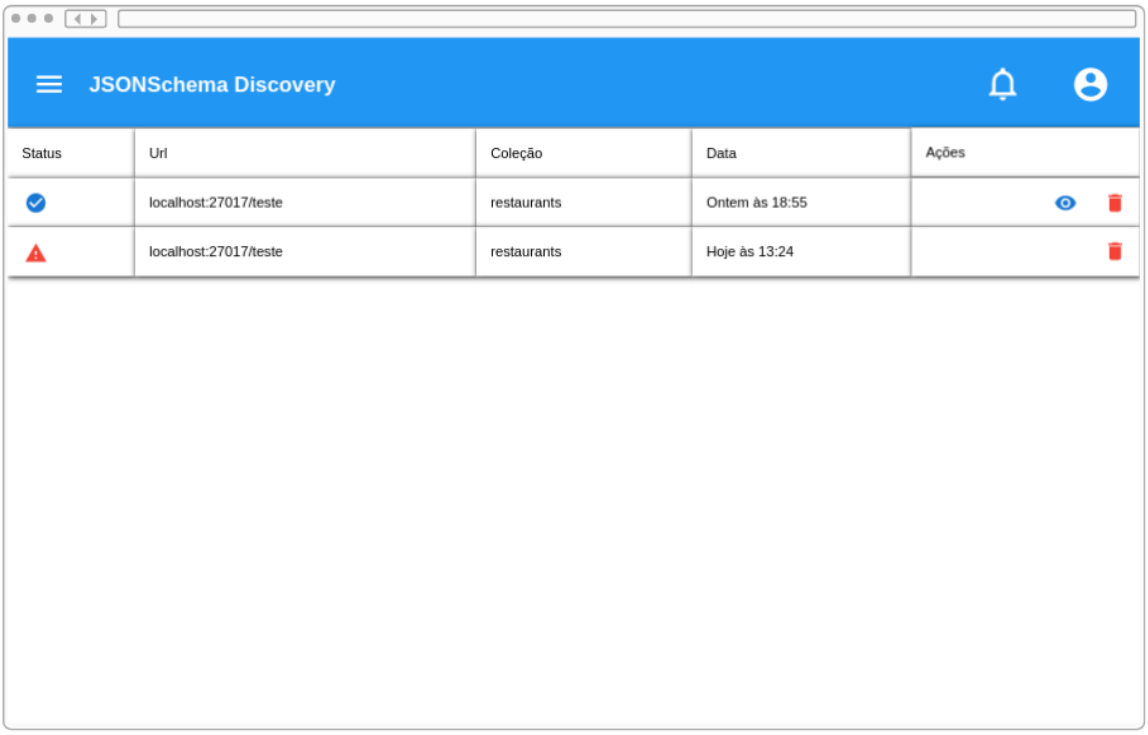
4.3 Opção sair (RF03)



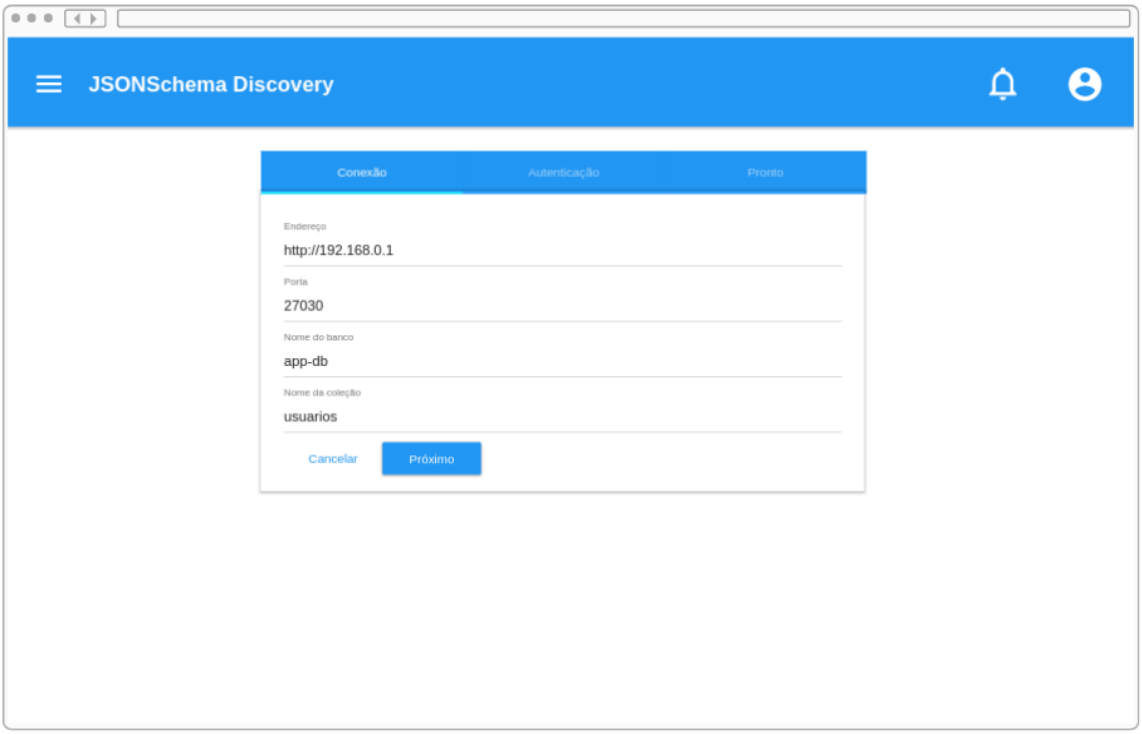
4.4 Workspace - Lista de esquemas (RF08, RF07)



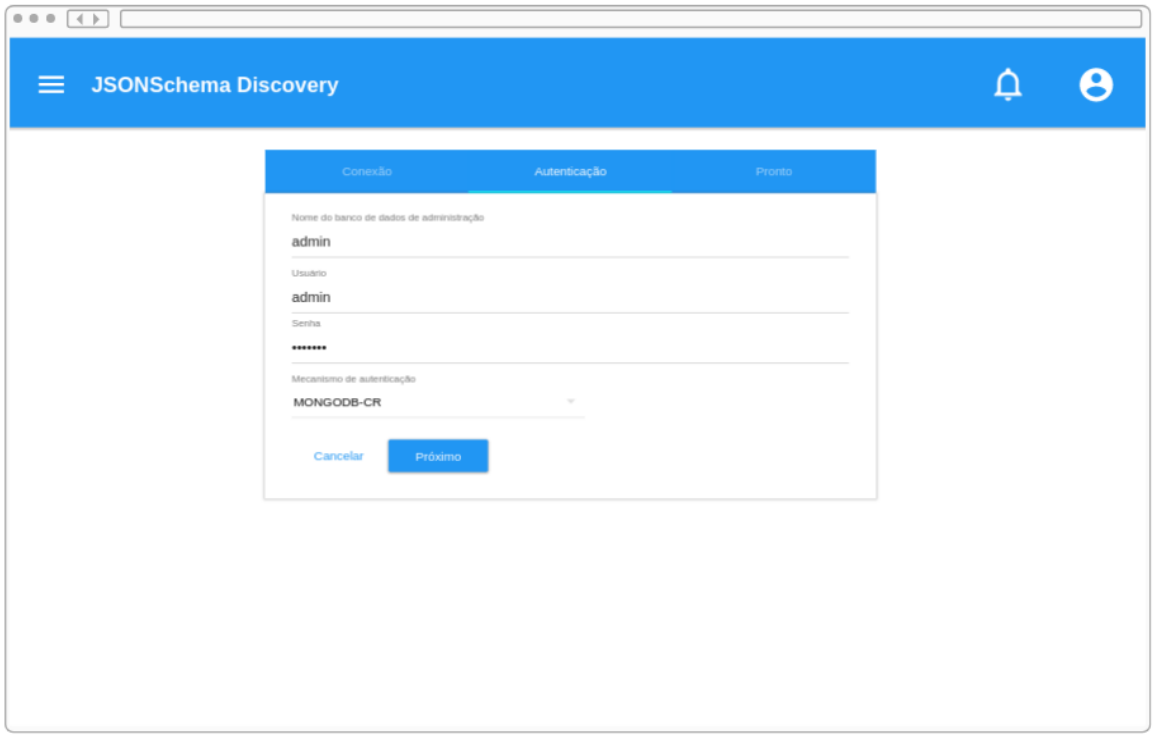
4.5 Área de alertas (RF09)



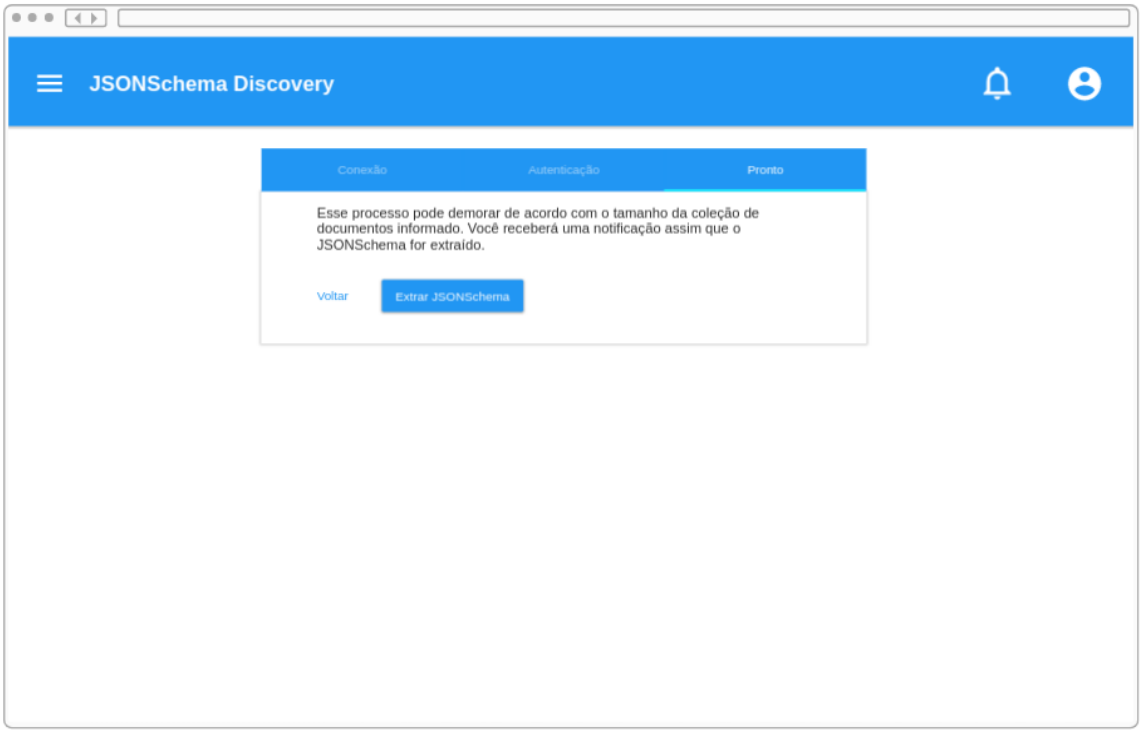
4.6 Extrair esquema – Dados de conexão (RF04)



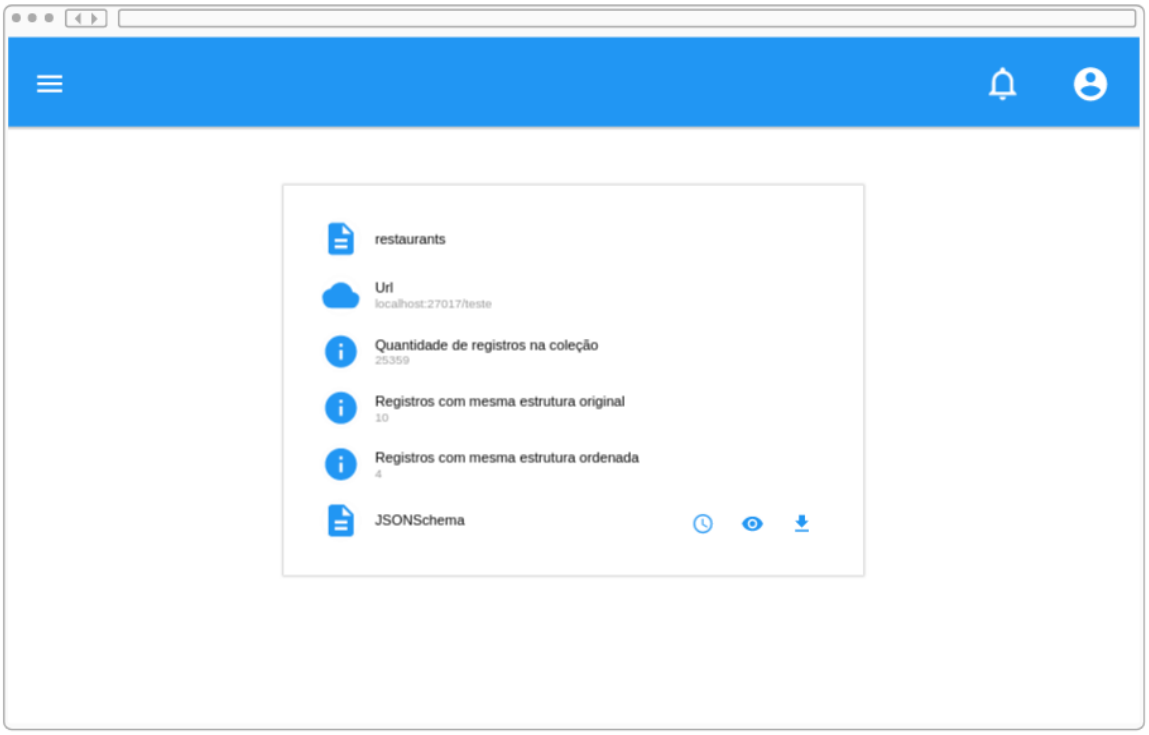
4.7 Extrair esquema – Dados de permissão (RF04)



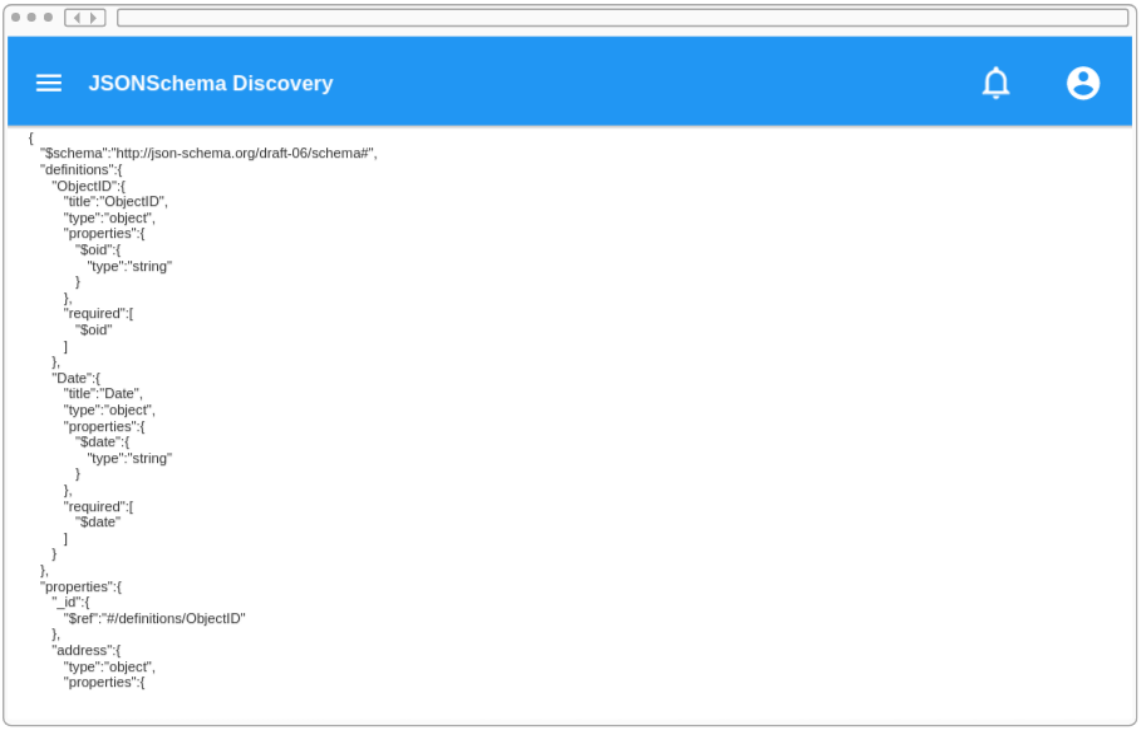
4.8 Extrair esquema – Confirmar (RF04)



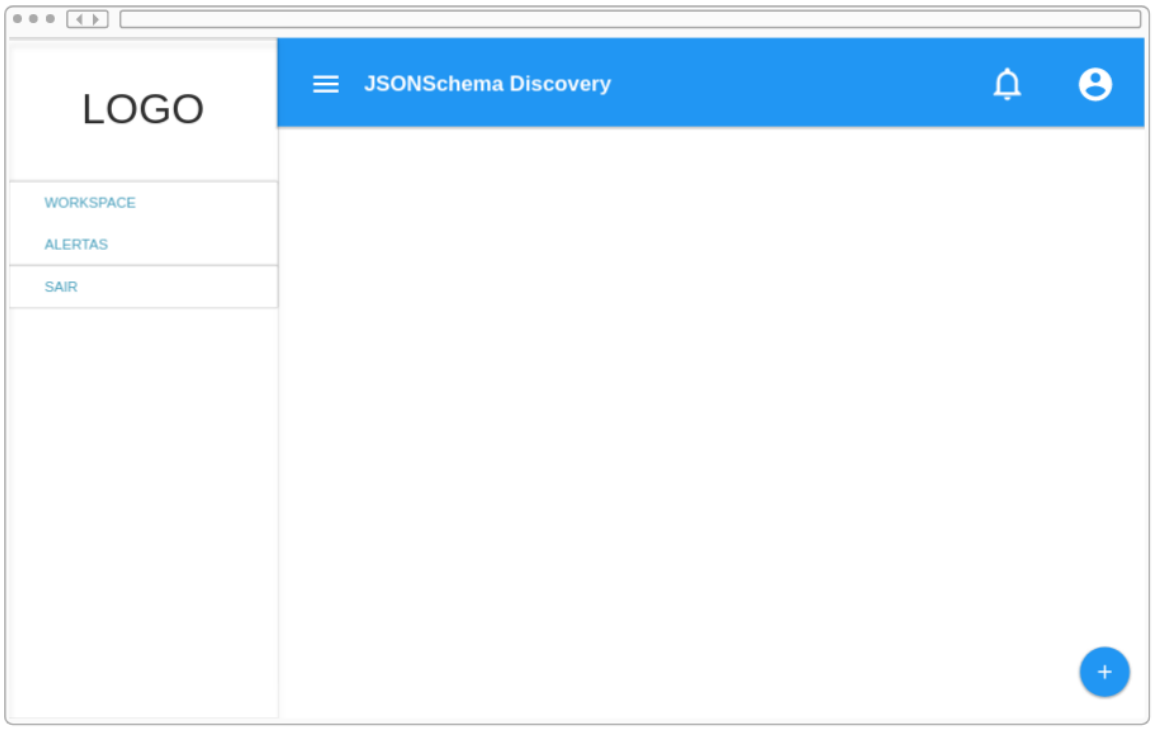
4.9 Detalhes do resultado (RF06, RF07)



4.10 Visualização do JSON Schema (RF06)



4.11 Navbar



APÊNDICE B – Artigo

Uma Ferramenta para Extração de Esquemas de Banco de Dados NoSQL Orientado a Documentos

Felipe de Souza da Costa[1]

[1]Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.049-900 – Florianópolis – SC – Brasil

felipe.souza.costa@grad.ufsc.br

Abstract. *NoSQL databases have become increasingly popular in application development, among other features, because of their ability to handle big data and and also their ability to be schemaless. Although most NoSQL databases are schemaless, information about them is essential during the application development. Thus, this work proposes a tool that extracts the schema from a collection of documents in JSON format, stored in a document-oriented NoSQL database, with the purpose of facilitating further data manipulation tasks, like data retrieval, integration, validation and analysis.*

Resumo. *Os bancos de dados NoSQL têm se tornando cada vez mais populares no desenvolvimento de aplicações, entre outras características, devido à sua capacidade de lidar com grandes volumes de dados e pela ausência de um esquema de dados explícito. Embora a maioria dos BDs NoSQL não tenha esquema, as informações sobre os mesmos são essenciais durante o desenvolvimento de aplicações. Sendo assim, o presente trabalho propõe o desenvolvimento de uma ferramenta que extraia o esquema de uma coleção de documentos no formato JSON, armazenados em um BD NoSQL orientado a documentos, com o objetivo de facilitar diversas tarefas de manipulação posterior desses dados, como a recuperação, validação, integração e análise de dados.*

1. Introdução

A explosão do mercado de *Big Data* fez com que grandes empresas demandassem por banco de dados (BDs) capazes de armazenar e processar grandes volumes de dados de forma eficaz. Para atender essas necessidades, uma variedade de sistemas de BDs surgiu, com o propósito de facilitar a expansão, apoiar o armazenamento em massa, suportar eficientemente operações de leitura e escrita e, além de tudo, ter um baixo custo de operação e manutenção. A esses BDs é dado o nome de BD NoSQL. Uma característica muito importante de BDs NoSQL é o fato de serem *schemaless*, ou seja, permitirem o armazenamento de dados sem o conhecimento prévio da estrutura dos mesmos [Sadalage and Fowler 2012, Ruiz et al. 2015]

Os BDs NoSQL são classificados em quatro categorias amplamente utilizadas: chave-valor, orientado a documento, orientado a colunas e orientado a grafos. Os BDs NoSQL orientados a documentos, por exemplo, armazenam e recuperam documentos, os quais podem ser nos formatos XML[1] (*eXtensible Markup Language*),

[1]https://www.w3.org/XML/

JSON[2] (*JavaScript Object Notation*) ou BSON[3] (*Binary* JSON) [Hashem and Ranc 2016, Indrawan-Santiago 2012, Han et al. 2011]. Em geral, não possuem esquema, ou seja, os documentos não precisam possuir uma estrutura em comum.

Os BDs NoSQL permitem que sejam armazenados quaisquer dados sobre uma chave ou uma coluna escolhida. Além disso, permitem que sejam adicionados, livremente, campos aos registros do BD, sem ter de definir primeiro quaisquer mudanças na estrutura, como ocorre nos BDs relacionais [Sadalage and Fowler 2012]. A ausência de esquema evita o problema de registros com vários elementos sem valor, quando estes não são uniformes (cada registro possui um conjunto diferente de dados), permitindo que cada registro contenha apenas o necessário — nada a mais, nada a menos. No entanto, a ausência de um esquema explícito, não acarreta a ausência total de esquema, visto que sempre há um esquema implícito nos dados e em aplicações de BDs [Ruiz et al. 2015].

Sendo assim, o objetivo deste trabalho é o desenvolvimento de uma ferramenta que, dada uma coleção de documentos no formato JSON (formato padrão para intercâmbio de dados complexos e heterogêneos [IETF 2014]), armazenados em um BD NoSQL orientado a documentos, extraía o esquema implícito na coleção, facilitando assim o desenvolvimento de sistemas, a integração e validação de dados, bem como a análise dos mesmos.

Estrutura. Este artigo está organizado da seguinte forma: A próxima seção fornece uma visão geral sobre o formato de dados JSON e a linguagem de especificação de esquemas JSON. Na seção 3 é apresentado os trabalhos relacionados. A proposta é apresentada na seção 4. Os resultados dos estudos de casos são fornecidos na seção 5. Finalmente, conclusões e trabalhos futuros são apresentados na seção 6.

2. Fundamentação teórica

2.1. JSON

JSON é um formato leve utilizado para intercâmbio de dados complexos e heterogêneos. Por ser um formato de texto completamente independente de linguagem [ECMA-404 2014], utilizando convenções que são familiares àquelas utilizadas em linguagens como C, C++, C#, *Java*, *JavaScript*, *Perl* e *Python*, o formato JSON foi amplamente adotado como um formato ideal para intercâmbio de dados.

JSON é construído basicamente em duas estruturas, presentes em muitas linguagens de programação:

- Uma coleção de pares chave-valor, semelhante a um objeto.
- Uma lista ordenada de valores, semelhante a um *array*.

A terminologia que define a estrutura de um documento JSON compreende os seguintes conceitos:

- *Objeto*: conjunto desordenado de pares chave-valor. Um objeto começa com '{' (abre-chaves) e termina com '}' (fecha-chaves), cada chave é seguida de ':' (dois-pontos) e os pares chave-valor são separados por ',' (vírgula);

[2] `http://json.org/`
[3] `http://bsonspec.org/`

- *Array*: coleção ordenada de valores. Um *array* começa com '[' (abre-colchetes) e termina com ']' (fecha-colchetes) e os valores são separados por ',' (vírgula);
- *Valor*: um valor JSON pode ser um valor de tipo primitivo (*string*, numérico, booleano (*true*, *false*)), de tipo estruturado (objeto, *array*) ou nulo *null*). O uso de objetos e *arrays* como valores permite definir estruturas aninhadas.

2.2. BSON

BSON[4], abreviação de *Binary* JSON, é uma serialização codificada em binário de documentos JSON. Como JSON, BSON suporta a incorporação de documentos e *arrays* em outros documentos e *arrays*. BSON também é considerado uma extensão do JSON que permitem a representação de tipos de dados que não fazem parte da especificação JSON. Por exemplo, BSON possui os tipos *Date*, *Timestamp*, *Binary*, *ObjectID*, *RegExp*, *Long*, *Undefined*, *DBRef*, *Code*, *MinKey* e *MaxKey*.

2.3. JSON *Schema*

O JSON *Schema*[5] é uma tentativa de fornecer uma linguagem de esquema de propósito geral para JSON, que permita aos usuários restringir a estrutura dos documentos JSON — nomes necessários em objetos, os tipos permitidos para um atributo de um objeto ou para valores em um *array*, além de oferecer uma estrutura para verificar a integridade das solicitações e sua conformidade com uma API [Pezoa et al. 2016].

A Figura 1(a) representa o corpo de uma requisição HTTP (*body*), no formato JSON, a uma API de um *web service*, que provê informações hipotéticas sobre restaurantes ao redor do mundo. A documento JSON, contido no corpo da requisição, possui as propriedades *latitude* e *longitude*, das quais se deseja obter informações.

A Figura 1(b) representa o JSON *Schema* para validar o corpo das requisições a API. Todas as requisições a API devem informar um documento JSON, com os atributos *latitute* e *longitude*, ambos do tipo *number* (*"type":"number"*), sendo que não é permitido atributos adicionais na requisição (*"additionalProperties":false*), além de ambos atributos serem obrigatórios (*"required":["latitude","longitude"]*).

```
{
  "latitude": -73.9574128,
  "longitude": 40.7701235
}
```

(a) Corpo de uma requisição HTTP

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "properties": {
    "latitude": {
      "type": "number"
    },
    "longitude": {
      "type": "number"
    }
  },
  "additionalProperties": false,
  "required": [
    "latitude",
    "longitude"
  ]
}
```

(b) JSON *Schema*

Figura 1. Documento JSON Estendido e Esquema bruto

[4] http://bsonspec.org/
[5] http://json-schema.org/

3. Trabalhos relacionados

O artigo de [IMHOF et al. 2017] intitulado "Um *Survey* sobre Extração de Esquemas de Documentos JSON", apresenta uma análise comparativa entre trabalhos relacionados ao tema de extração de esquemas de documentos JSON. Este artigo foi utilizado como referência para a definição dos principais trabalhos relacionados à ferramenta proposta e facilitou na identificação das características de cada trabalho.

A Tabela 1 apresenta um comparativo das principais características observadas nos trabalhos relacionados. A última coluna se refere a este trabalho, que é também comparado com os demais. As características consideradas foram as seguintes: (*i*) Origem (fonte de origem usada para obter documentos JSON); (*ii*) Objetivo (perspectiva de uso dos esquemas extraídos); (*iii*) Abordagem (abordagem utilizada para obter o esquema final); (*iv*) Formato de entrada suportado (se suporta mais de um formato de entrada); (*v*) Modelo Intermediário ; (*vi*) Modelo de saída (como o esquema é disponibilizado ao final do processo); e (*vii*) Etapas do processo (um resumo das etapas do processo de extração adotado).

Tabela 1. Comparação entre propostas para extração de esquemas de BDs NoSQL

	[Klettke et al. 2015]	[Ruiz et al. 2015]	[Wang et al. 2015]	[Izquierdo and Cabot 2013]	[Wischenbart et al. 2012]	(Costa, 2017)
Origem dos dados	*Dataset* no *MongoDB*	*MongoDB*, *CouchDB* e *HBase*	*Datasets* JSON	APIs de serviços *web*	APIs de redes sociais	*Dataset* no *MongoDB*
Objetivo	Ferramenta para visualização e validação de esquemas.	Ferramenta para visualização e validação de esquemas	*Framework* para consulta e visualização de esquemas	Criar visão de domínio para os serviços da API	Integrar perfis de usuário em redes sociais	Ferramenta *web* para extração e visualização de esquemas.
Abordagem	Organização hierárquica (grafo)	Transformação de modelos	Organização hierárquica (árvore)	Transformação de modelos	Transformação de modelos	Organização hierárquica e Transformação de modelos
Formato entrada	JSON	JSON	JSON	JSON	JSON	JSON e JSON Estendido
Modelo Interm.	*Structure Identification Graph* (SG)	Metamodelo JSON	*eSiBu-Tree*	Metamodelo JSON (ECORE)	JSON *Schema*	*Raw Schema Unified Structure* (RSUS)
Modelo Saída	JSON *Schema*	Metamodelo de esquema JSON	*eSiBu-Tree*	Modelo de domínio da Aplicação (ECORE)	Modelo de classes (ECORE)	JSON *Schema*
Tipo de software	Algoritmo	Algoritmo	*Framework*	*Plugin Eclipse*	Algoritmo	SaaS (Software-as-a-Service)
Etapas	a) Seleção de documentos JSON; b) Extração da estrutura dos documentos; c) Construção do grafo SG; d) Geração do JSON *Schema*.	a) Extração da estrutura dos dados JSON (*raw schema*) ; b) Transformação para esquemas JSON; c) Transformação para esquema NoSQL.	a) Seleção de documentos JSON; b) Criação de registro na *eSiBu-Tree*; c) Visualização do esquema unificado (*skeleton*).	a) Pré-descoberta de documentos JSON; b) Extração de esquema do serviço; c) Criação do esquema de domínio.	a) Extração de dados; b) Extração de esquemas; c) Transformação; d) Integração.	a) Extração da estrutura dos documentos); b) Agrupação de estruturas extraídas; c) Construção da RSUS; d) Geração do JSON *Schema*.

Este capítulo apresentou uma análise dos trabalhos similares à ferramenta proposta. Percebe-se que ambos os trabalhos compartilham um objetivo geral em comum: manipulação e gerenciamento de esquemas. No entanto, a ferramenta proposta se destaca das demais por ser uma aplicação *web* e oferecer uma interface gráfica intuitiva para o

usuário. A ferramenta também mantém o histórico dos esquemas extraídos e é a única ferramenta para este fim no mercado. Destaque também para o formato de dados de entrada suportado e para o modelo de saída. O presente trabalho, bem como a proposta de [Klettke et al. 2015], utilizam como modelo de saída uma especificação padrão para definição de esquemas JSON (JSON *Schema*).

Os diferenciais dessa ferramenta quanto aos trabalhos relacionados são:

- SaaS (*Software-as-a-Service*) — acesso via qualquer *browser*;
- Persistência dos esquemas gerados;
- Histórico de esquemas gerados;
- Visualização e *download* de esquemas gerados;
- Suporte a JSON Estendido.

4. JSONSchema Discovery

4.1. Obtendo os esquemas brutos dos documentos

O primeiro passo para extrair o JSON *Schema* é obter o esquema bruto (*raw schema*) de cada documento JSON de uma coleção, passo semelhante ao apresentado por [Ruiz et al. 2015], e armazenar os resultados em uma coleção temporária do *MongoDB*. O esquema bruto contém a mesma estrutura que o documento JSON original com relação à campos, objetos aninhados e *arrays*. Cada valor primitivo no documento é substituído no esquema bruto por seu tipo de dado JSON (por exemplo, *string* ou *number*). Um diferencial nesta proposta é o fato de considerar os tipos de dados presentes no JSON Estendido do *MongoDB* (por exemplo: *date*, *objectId*, entre outros).

O Algoritmo 1 realiza a extração do esquema bruto de cada documento, dada uma coleção de documentos JSON como entrada. Como saída, é retornada uma coleção de esquemas brutos.

Algoritmo 1: Operação de extração do *raw schema* dos documentos

```
1  function Parser(collection);
   Input  : collection of documents JSON
   Output: rawSchemes of documents JSON
2  begin
3      rawSchemes ← ∅;
4      for document ∈ collection do
5          documentRawSchema ← {};
6          for key ∈ keys(document) do
7              value ← document[key];
8              documentRawSchema[key] ← BuildRawSchema(value);
9          end
10         add documentRawSchema to rawSchemes;
11     end
12     return rawSchemes;
13 end
```

O Algoritmo 2 descreve a função *BuildRawSchema* invocada no Algoritmo 1. Ela recebe um valor como entrada e realiza a extração do esquema bruto do valor. Se o tipo

do valor for um tipo JSON primitivo ou um tipo JSON Estendido, o algoritmo retorna o nome do tipo. Caso contrário, é aplicada recursão para valores do tipo objeto ou *array*.

Algoritmo 2: Composição do *raw schema*

```
1  function BuildRawSchema(value);
   Input  : value of key-value pair
   Output: rawSchema of value
2  begin
3      instance ← NULL;
4      if extendedJSONTypeOf(value) ≠ NULL then
5          instance ← extendedJSONTypeOf(value);
6      else if value = UNDEFINED then
7          instance ← "undefined";
8      else if value = NULL then
9          instance ← "null";
10     else if typeof(value) = Array then
11         instance ← ∅;
12         for item ∈ value do
13             itemRawSchema ← BuildRawSchema(item);
14             add itemRawSchema to instance;
15         end
16     else if typeof(value) = Object then
17         instance ← {};
18         for key ∈ keys(value) do
19             keyValue ← value[key];
20             instance[key] ← BuildRawSchema(keyValue);
21         end
22     else
23         instance ← typeof(value);
24     end
25     return instance;
26 end
```

4.2. Agrupando esquemas brutos iguais

Na segunda etapa, duas operações de agregação são aplicadas para extrair uma coleção de objetos JSON únicos, ou seja, o número mínimo de objetos necessários para executar o processo de unificação dos esquemas brutos. A primeira operação de agregação é aplicada na coleção resultante da primeira etapa com o objetivo de agrupar os documentos que possuem o mesmo esquema bruto. O resultado dessa operação é armazenado em uma coleção temporária que é utilizada pela segunda operação de agregação.

Antes de aplicar a segunda operação de agregação, é realizada a ordenação alfabética das propriedades dos esquemas brutos resultantes da primeira etapa de agregação. Considerando que os elementos de um documento JSON não precisam seguir a mesma ordem que estão definidos no esquema, o objetivo desta ordenação é reduzir ainda mais

a quantidade de documentos com esquemas brutos distintos. Após a ordenação ser concluída, é iniciada a segunda operação de agregação que resulta na menor quantidade de documentos com esquema bruto distintos.

4.3. Unificação de Esquemas Brutos

Na terceira etapa ocorre a unificação dos esquemas brutos resultantes da etapa 2. Para a unificação de esquemas brutos foi necessário definir uma estrutura de dados temporária, que armazenasse de forma performática, informações sobre a estrutura hierárquica de cada esquema bruto, sendo essas informações: os atributos dos objeto e seus tipos de dados, os itens dos *arrays* e seus tipos de dados, caminho da raiz do documento até uma propriedade ou item qualquer, bem como a quantidade de ocorrências para cada propriedade, com base no caminho e no tipo de dado, nos documentos de uma coleção. Assim, as informações de todos os esquemas brutos são armazenados na chamada Estrutura Unificada de Esquema Bruto ou, em Inglês, ***R**aw **S**chema **U**nified **S**tructure* (RSUS).

As propriedades de cada esquema bruto são percorridas em *preorder* e, para cada propriedade, é realizada uma adição ou extensão de uma propriedade na RSUS. Caso a propriedade represente um atributo de objeto, então é realizado uma adição ou extensão de uma propriedade na RSUS. Toda adição ou extensão, leva em conta o caminho da raiz do esquema bruto até a propriedade que está sendo analisada.

4.4. Obtendo o JSON *Schema*

Uma vez realizada a unificação dos esquemas brutos, um algoritmo baseado em transformação de modelos é empregado, para transformar a estrutura RSUS em um JSON *Schema* válido. A ideia geral dessa etapa é gerar um JSON *Schema* para cada propriedade na RSUS (Algoritmo 3), respeitando a estrutura hierárquica do esquema bruto.

Algoritmo 3: Mapeamento RSUS $\rightarrow$ JSON *Schema*

```
1 function getFieldsSchema(fields);
  Input  : fields
  Output: properties
2 begin
3 |   properties ← {};
4 |   for field ∈ fields do
5 |   |   properties[field.name] ← getSchemaFromValue(field);
6 |   end
7 |   return properties;
8 end
```

5. Resultados alcançados

A avaliação considera três cenários: (A) *qualitativo* – a fim de verificar a qualidade dos mapeamentos JSON $\rightarrow$JSON *Schema*; (B) *tempo de processamento* – a fim de analisar o tempo de processamento da ferramenta; (C) *comparação com trabalhos relacionados* – a fim de comparar os resultados com trabalhos relacionados. Estes cenários são descritos a seguir.

5.1. Avaliação qualitativa

Esta avaliação visa verificar a corretude e completude dos mapeamentos JSON → JSON *Schema*. Para isso, foram criados cinco documentos JSON hipotéticos, com atributos representando todos os tipos de dados JSON e JSON Estendido, ordenação das propriedades diferentes entre os documentos, bem como, diferença de tipos para uma mesma propriedade entre os documentos. Esses documentos foram armazenados em uma coleção no *MongoDB*.

A Figura 2(a) apresenta o exemplo de um dos documentos JSONs deste experimento, possuindo tipos JSON Estendido, bem como um *array* com itens de tipos diferentes. A Figura 2(b) apresenta o esquema bruto do documento obtido na primeira etapa do processo.

```
{
  "_id": ObjectId("50319491fe4dce143835c555"),
  "stringType": "some_value",
  "dateAsTime": { "$date":
      { "$numberLong": "1496795859332" }
  },
   "maxKeyType": { "$maxKey": 1 },
  "minKeyType": { "$minKey": 4 },
  "intType":1,
  "doubleType": 12343.44,
  "intType2": { "$numberInt": "1" },
  "objectType": { "property": "some_value" },
  "arrayMultiValues": [ 48, "50",
      {"property": "teste"},
      [1,"some_value"],
      true
  ],
  "arrayOfString": [ "some_value", "some_value" ],
  "arrayOfNumber": [ ],
  "arrayOfBoolean": [ true, false ],
  "arrayOfArray": [
      ["some_value", "some_value"]
  ],
  "arrayOfExtendedType": [
      Date("2014-01-31T22:26:33.000Z"),
      NumberLong("9223372036854775807")
  ]
}
```

(a) Documento JSON Estendido

```
{
  "_id": "ObjectID",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ]
}
```

(b) Esquema bruto

Figura 2. Documento JSON Estendido e Esquema bruto

A segunda etapa do processo tem como objetivo obter um documento para cada esquema bruto distinto. Para isto, é aplicado duas operações de agregação, sendo a primeira aplicada na coleção de documentos resultantes da primeira etapa do processo. O resultado é uma redução significativa da quantidade de documentos com um mesmo esquema bruto. Como a estrutura de um objeto JSON é constituído por uma lista de pares chave-valor não ordenados, pode haver dois documentos com as mesmas propriedades, porém em ordens distintas. Sendo assim, é realizado uma ordenação alfabética da estrutura destes documentos, para então se aplicar uma nova operação de agregação, que resultará em uma quantidade ainda menor de documentos com esquema bruto distintos.

Na terceira etapa, os documentos resultantes da etapa anterior são utilizados na construção da RSUS. A RSUS é um documento JSON e, como tal, pode ser visualizado como uma árvore hierárquica (Figura 3). Não foi possível representar a estrutura completa

devido a limites de espaço, no entanto, as propriedades mais complexas são apresentadas. Os números presentes nas propriedades são utilizados para definir se a propriedade é obrigatória ou opcional.

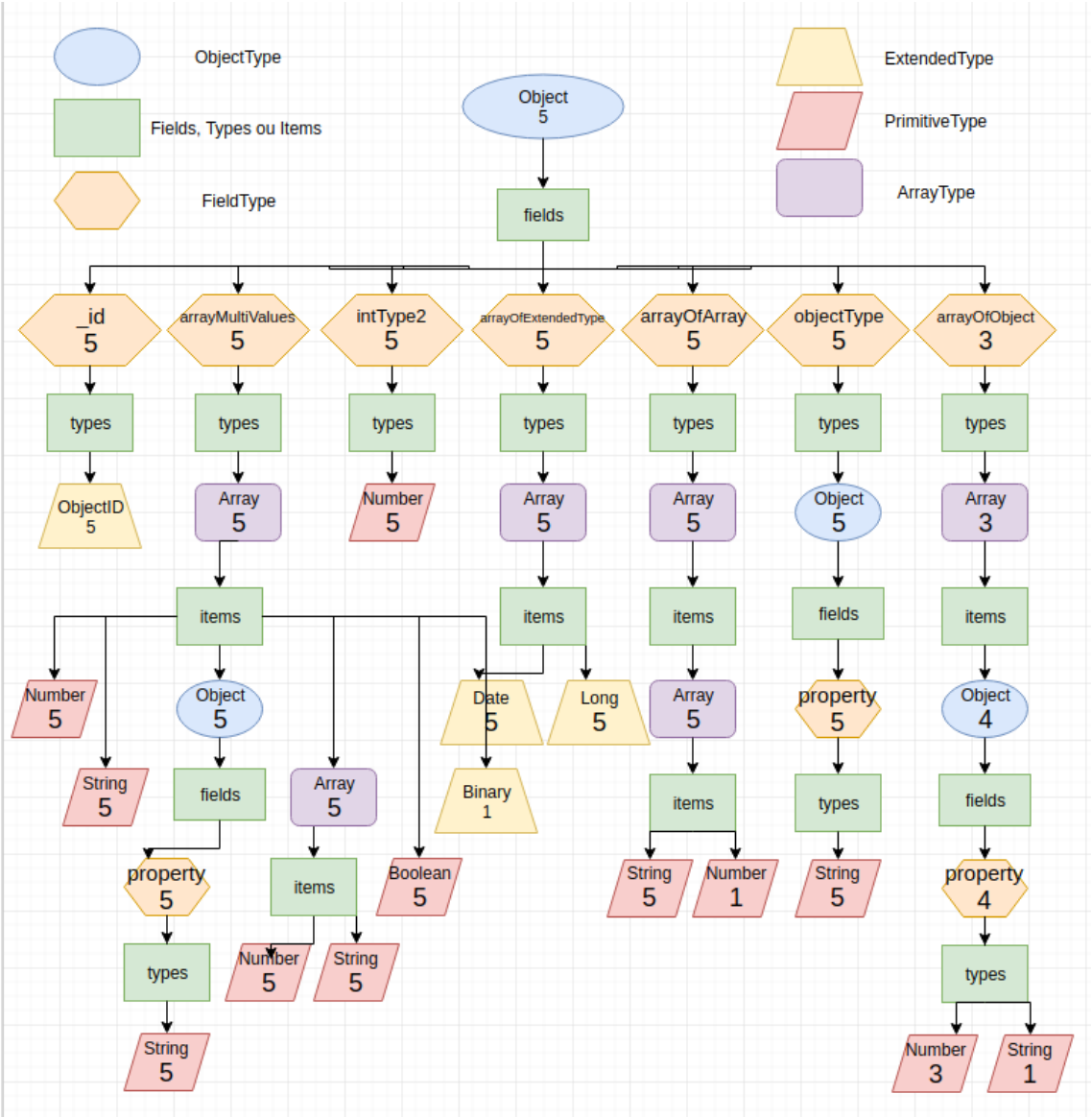


Figura 3. Visualização da RSUS como uma árvore hierárquica

Por fim, na quarta etapa há o mapeamento RSUS → JSON *Schema*. Não foi possível apresentar o resultado completo devido a limites de espaço, no entanto, alguns fragmentos do JSON *Schema* gerado são apresentados nas Figuras 4(a) e 4(b).

Quanto a qualidade do esquema esquema gerado, obteve-se uma precisão de 100%, visto que foram identificados todos os tipos de dados esperados nos documentos. Estes tipos de dados incluem atributos obrigatórios, tipos JSON Estendido, quantidade de itens mínimos em um *array* e tipos de união para quando uma propriedade possui mais de um tipo de dado.

(a) JSON *Schema* (fragmento)

(b) JSON *Schema* (fragmento)

Figura 4. JSON *Schema* gerado

5.2. Análise do tempo de processamento

Nesta seção são apresentados os resultados da análise de tempo de processamento da extração de esquemas para coleções massivas de dados (*massive datasets*). Os experimentos foram executados em uma instância *Amazon* EC2 *t2.micro* (Intel® Xeon® E5-2676 v3 @ 2.40GHz e 1GB de memória RAM).

Os *datasets* utilizados no experimento foram obtidos de um trabalho existente [Çelikten et al. 2017] e correspondem a dados rastreados de *"tweets"*, *"checkins"* e locais (*"venues"*) do *Foursquare*. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

Os resultados experimentais da Tabela 2 mostram o tempo de processamento gasto em média no processo de extração de esquema para os três *datasets*. Vale destacar a importância da etapa de ordenação das propriedades do esquema bruto, como observado para o *dataset "venues"*, em que a quantidade real de esquemas brutos distintos era de 117, ao invés de 257 obtidos antes da ordenação das propriedades. Vale ressaltar, também, o tempo de processamento gasto para leitura dos documentos e geração dos esquemas brutos, chegando a 99% do tempo total do processo, visto que é realizado a leitura de todos os documentos de uma coleção. Várias técnicas foram utilizadas e testadas nesta etapa, como o uso de *streams* e processamento assíncrono, visando a otimização do tempo total. Esse tempo pode ser melhorado, em trabalhos futuros, melhorando a configuração da máquina de testes e usando *clusters* para processamento paralelo.

Tabela 2. Resultados para os *datasets venues*, *tweets* e *checkins*

Coleção	Número de documentos	Esquema bruto	Esquema bruto com estrutura ordenada	Tempo obtenção dos esquemas brutos (TB)	TB/TT	Tempo total (TT)
venues	2.096.353	257	117	7,47 min	99,33%	7,52 min
checkings	11.577.228	2	2	35,27 min	99,29%	35,52 min
tweets	16.972.245	23	16	53,11 min	99,38%	53,44 min

5.3. Comparação com trabalhos relacionados

Nesta seção são apresentados os resultados da análise do processo de extração de esquemas para *datasets* utilizados em trabalhos relacionados. Os experimentos foram executados em um *notebook* (*Intel® Core*™ i7 3610QM @ 2.30 GHz e 8GB de memória RAM).

Os *datasets* (*drugs*, *companies* e *movies*) utilizados no primeiro experimento (tabela 3) foram obtidos do trabalho de [Wang et al. 2015] e correspondem a dados retirados da *DBPedia*[6]. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

Quanto a quantidade de esquemas distintos identificados, ambas propostas chegaram a mesma quantidade para o *dataset drugs*, enquanto que para os *datasets companies* e *movies*, o presente trabalho identificou uma quantidade maior. Essa diferença ocorreu, pois ao contrário da proposta de [Wang et al. 2015], a presente proposta leva em

[6] `http://wiki.dbpedia.org/`

consideração o nome e tipo de cada propriedade de um documento para montar seu respectivo esquema bruto, e não apenas os nomes das propriedades, como é realizado na criação do esquema de registro, utilizado no processo proposto por [Wang et al. 2015]. O tipo do dado também é importante para permitir que se faça consultas mais exatas em relação a filtros sobre determinados atributos. Por exemplo, se o tipo de dado de um atributo é numérico, o usuário poderá definir um filtro envolvendo uma constante numérica.

Tabela 3. Resultados para os *datasets drugs*, *companies* e *movies*

Datasets DBPedia		JSONSchema Discovery		WANG et. al. (2015)
Coleção	Número de documentos	Esquemas brutos	Esquemas brutos com estrutura ordenada	Esquemas
drugs	3662	2818	2818	2818
companies	24367	21312	21312	21302
movies	30330	25140	25140	25137

O *dataset* (*pullrequest*) utilizado no segundo experimento (Tabela 4) foi obtido de [Baazizi et al. 2017] e corresponde a dados retirados do *GitHub*[7]. Os mesmos estão no formato JSON e armazenados em um BD de documentos.

O *dataset* do *GitHub* corresponde a metadados gerados com *pull requests* emitidos por usuários. Três testes foram realizados, selecionando mil, dez mil e cem mil documentos, para se comparar a quantidade de esquemas identificados em cada teste. Em ambas as propostas foram identificados a mesma quantidade de esquemas.

Tabela 4. Resultados para o *dataset pullrequests*

Dataset GitHub		JSONSchema Discovery		Baazizi et. al. (2017)
Coleção	Número de documentos	Esquemas brutos	Esquemas brutos com estrutura ordenada	Esquemas
pullrequests	1000	29	29	29
pullrequests	10000	66	66	66
pullrequests	100000	261	261	261

6. Conclusão

Este trabalho teve como objetivo principal o desenvolvimento de uma ferramenta *web* para extração de esquemas de documentos JSON armazenados em um BD orientado a documentos. A principal motivação para este trabalho é facilitar tarefas como a manipulação e a integração de dados, pois com um esquema explicito definido, tem-se o conhecimento geral das propriedades obrigatórias e opcionais existentes em uma coleção de documentos, bem como, o tipo de dado permitido em cada propriedade. A definição de um esquema também pode ser utilizado na validação de dados e facilita também no desenvolvimento de *software*, visto que restrições de esquema não precisam ser especificadas dentro da lógica da aplicação.

A *JSONSchema Discovery* destaca-se das demais propostas pelas seguintes razões: é uma ferramenta *web*, estando, deste modo, disponível a qualquer dispositivo

[7] `https://github.com/`

com acesso a um navegador; utiliza uma especificação padrão para definição de esquemas de documentos JSON (JSON *Schema*); armazena os esquemas gerados; mantém o histórico dos esquemas gerados; permite a visualização do JSON *Schema* gerado; oferece suporte à JSON estendido; e apresenta uma interface gráfica simples e intuitiva, além de ser totalmente gratuita.

Como essa ferramenta foi concebida para extrair esquemas de documentos JSON armazenados no SGBD *MongoDB*, já se assume que os documentos estão bem formados. Entretanto, para o caso dessa ferramenta ser estendida para extrair esquemas de qualquer fonte de dados JSON, sua limitação é não ser capaz de extrair esquemas de documentos JSON mal formados. Pretende-se tratar essa limitação em trabalhos futuros, conforme descrito a seguir.

Algumas funcionalidades podem ser adicionadas, a fim de agregar maior maturidade à ferramenta, possibilitando assim sua melhoria contínua. Dessa forma, apresentam-se as seguintes possibilidades de trabalhos futuros:

- Melhorar a *performance* da primeira etapa de extração;
- Geração de esquemas para outros SGBDs NoSQL;
- Permitir a opção de *upload* de um ou mais documentos JSON para extração, considerando a existência de um *parser* que verifica se os documentos estão bem formados;
- Identificar e tratar relacionamentos por referência;
- Internacionalização do *software*, gerando versões em língua inglesa e espanhola.

Os *links* para a ferramenta podem ser encontrados no repositório *GitHub*[8] da aplicação, no qual é possível encontrar também todo o código, fazer sugestões e até colaborar com novas implementações.

Referências

Baazizi, M.-a., Pierre, U., Lahmar, H. B., Colazzo, D., Ghelli, G., Sartiani, C., Basilicata, U., and Informatica, D. (2017). Schema Inference for Massive JSON Datasets. *Edbt*, pages 222–233.

ECMA-404 (2014). The JavaScript Object Notation (JSON) Data Interchange Format. *ECMA International*, 1st Editio(October):8.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on NoSQL database. *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, pages 363–366.

Hashem, H. and Ranc, D. (2016). Evaluating NoSQL document oriented data model. *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*, pages 51–56.

IETF (2014). RFC 7159 - *The JavaScript Object Notation (JSON) Data Interchange Format*.

IMHOF, R., FROZZA, A. A., and MELLO, R. d. S. (2017). Um Survey sobre Extração de esquemas de documentos json. *Escola Regional de Banco de Dados - ERBD 2017*, pages 26–35.

[8] `https://github.com/feekosta/JSONSchemaDiscovery`

Indrawan-Santiago, M. (2012). Database research: Are we at a crossroad? Reflection on NoSQL. *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012*, pages 45–51.

Izquierdo, J. L. C. and Cabot, J. (2013). Discovering implicit schemas in json data. In *Proceedings of the 13th International Conference on Web Engineering*, ICWE'13, pages 68–83, Berlin, Heidelberg. Springer-Verlag.

Klettke, M., Störl, U., and Scherzinger, S. (2015). Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In *Btw*, volume 241 of *LNI*, pages 425–444. GI.

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoč, D. (2016). Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, WWW '16, pages 263–273. International World Wide Web Conferences Steering Committee.

Ruiz, D. S., Morales, S. F., and Molina, J. G. (2015). Inferring Versioned Schemas from NoSQL Databases and its Applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9381(October):467–480.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*.

Wang, L., Zhang, S., Shi, J., Jiao, L., Hassanzadeh, O., Zou, J., and Wangz, C. (2015). Schema management for document stores. *Proceedings of the VLDB Endowment*, 8(9):922–933.

Wischenbart, M., Mitsch, S., Kapsammer, E., Kusel, A., Pröll, B., Retschitzegger, W., Schwinger, W., Schönböck, J., Wimmer, M., and Lechner, S. (2012). User profile integration made easy: Model-driven extraction and transformation of social network schemas. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 939–948, New York, NY, USA. ACM.

Çelikten, E., Falher, G. L., and Mathioudakis, M. (2017). Modeling urban behavior by mining geotagged social data. *IEEE Transactions on Big Data*, 3(2):220–233.

APÊNDICE C – Código fonte

```
# JSONSchemaDiscovery

## What you need installed to run this project:
* [NodeJS](http://nodejs.org)
* [Mongo DB](https://www.mongodb.org)

## Setting up development environment:
After clone the repo to your local machine, in project's folder:
1. Install global dependencies:
* [Angular CLI](https://cli.angular.io/) `npm install -g @angular/cli`
* [Typescript](https://www.typescriptlang.org/) `npm install -g typescript`

2. Install project dependencies running: `npm install`;

3. Configuring environment variables
create an .env file with the following global variables:
MONGODB_URI = mongodburl
SECRET_TOKEN = anytoken

## Development server

Run `npm run dev` for a dev server. Navigate to `http://localhost:4200/`. The app will automatically reload if you change any of the source files.
```

Arquivo 1: codigofonte/README.md

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "project": {
    "name": "JSONSchemaDiscovery"
  },
  "apps": [
    {
      "root": "client",
      "outDir": "dist/public",
      "assets": [
        "assets",
        "assets/favicon.ico"
      ],
      "index": "index.html",
      "main": "main.ts",
      "polyfills": "polyfills.ts",
      "test": "test.ts",
      "tsconfig": "tsconfig.app.json",
      "testTsconfig": "tsconfig.spec.json",
      "prefix": "app",
      "styles": [
        "../node_modules/bootstrap/dist/css/bootstrap.min.css",
        "../node_modules/font-awesome/css/font-awesome.min.css",
```

```
        "styles.css"
      ],
      "scripts": [
        "../node_modules/jquery/dist/jquery.min.js",
        "../node_modules/bootstrap/dist/js/bootstrap.min.js"
      ],
      "environmentSource": "environments/environment.ts",
      "environments": {
        "dev": "environments/environment.ts",
        "prod": "environments/environment.prod.ts"
      }
    }
  ],
  "e2e": {
    "protractor": {
      "config": "./protractor.conf.js"
    }
  },
  "lint": [
    {
      "project": "client/tsconfig.app.json",
      "exclude": "**/node_modules/**"
    },
    {
      "project": "client/tsconfig.spec.json",
      "exclude": "**/node_modules/**"
    },
    {
      "project": "e2e/tsconfig.e2e.json",
      "exclude": "**/node_modules/**"
    }
  ],
  "test": {
    "karma": {
      "config": "./karma.conf.js"
    }
  },
  "defaults": {
    "styleExt": "css",
    "component": {}
  }
}
```

Arquivo 2: codigofonte/.angular–cli.json

```
# Editor configuration, see http://editorconfig.org
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 2
insert_final_newline = true
trim_trailing_whitespace = true
```

```
10
11 [*.md]
12 max_line_length = off
13 trim_trailing_whitespace = false
```

Arquivo 3: codigofonte/.editorconfig

```
1  # See http://help.github.com/ignore-files/ for more about ignoring
       files.
2
3  # compiled output
4  /dist
5  /tmp
6  /out-tsc
7
8  # dependencies
9  /node_modules
10
11 # IDEs and editors
12 /.idea
13 .project
14 .classpath
15 .c9/
16 *.launch
17 .settings/
18 *.sublime-workspace
19 *.sublime-project
20
21 # IDE - VSCode
22 .vscode/*
23 !.vscode/settings.json
24 !.vscode/tasks.json
25 !.vscode/launch.json
26 !.vscode/extensions.json
27
28 # misc
29 /.sass-cache
30 /connect.lock
31 /coverage
32 /libpeerconnection.log
33 npm-debug.log
34 testem.log
35 /typings
36 yarn-error.log
37 crash.log
38
39 # e2e
40 /e2e/*.js
41 /e2e/*.map
42
```

```
# System Files
.DS_Store
Thumbs.db

# Logs folder
/logs
```

Arquivo 4: codigofonte/.gitignore

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/0.13/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular/cli'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular/cli/plugins/karma')
    ],
    client:{
      clearContext: false // leave Jasmine Spec Runner output visible
          in browser
    },
    coverageIstanbulReporter: {
      reports: [ 'html', 'lcovonly' ],
      fixWebpackSourcePaths: true
    },
    angularCli: {
      environment: 'dev'
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
```

Arquivo 5: codigofonte/karma.conf.js

```
{
  "name": "JSONSchemaDiscovery",
  "version": "0.0.1",
  "license": "MIT",
  "author": "Felipe de Souza da Costa",
  "description": "Web application for extraction of documents nosql
      database schemas",
```

```
  "angular-cli": {},
  "scripts": {
    "ng": "ng",
    "start": "node dist/server/app.js",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "predev": "tsc -p server",
    "dev": "concurrently \"mongod\" \"ng serve -pc proxy.conf.json --
       open\" \"tsc -w -p server\" \"nodemon dist/server/app.js\"",
    "prod": "concurrently \"ng build -aot -prod && tsc -p server &&
       node dist/server/app.js\"",
    "postinstall": "tsc -p server && ng build --aot -prod"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^4.3.6",
    "@angular/cdk": "^2.0.0-beta.10",
    "@angular/cli": "^1.4.3",
    "@angular/common": "^4.0.0",
    "@angular/compiler": "^4.0.0",
    "@angular/compiler-cli": "^4.0.0",
    "@angular/core": "^4.0.0",
    "@angular/forms": "^4.0.0",
    "@angular/http": "^4.0.0",
    "@angular/material": "^2.0.0-beta.11",
    "@angular/platform-browser": "^4.0.0",
    "@angular/platform-browser-dynamic": "^4.0.0",
    "@angular/router": "^4.0.0",
    "@types/node": "~6.0.60",
    "angular2-moment": "^1.7.0",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.17.2",
    "bootstrap": "^3.3.7",
    "core-js": "^2.4.1",
    "dotenv": "^4.0.0",
    "event-stream": "^3.3.4",
    "express": "^4.15.3",
    "file-saver": "^1.3.3",
    "font-awesome": "^4.7.0",
    "hammerjs": "^2.0.8",
    "jquery": "^3.2.1",
    "jsonwebtoken": "^7.4.2",
    "mongodb": "^2.2.30",
    "mongoose": "^4.11.5",
    "morgan": "^1.8.2",
    "rotating-file-stream": "^1.2.2",
    "rxjs": "^5.4.1",
    "segfault-handler": "^1.0.0",
    "ts-node": "~3.0.4",
    "typescript": "~2.3.3",
    "zone.js": "^0.8.14"
  },
```

```
  "devDependencies": {
    "@angular/language-service": "^4.0.0",
    "@types/jasmine": "~2.5.53",
    "@types/jasminewd2": "~2.0.2",
    "codelyzer": "~3.0.1",
    "concurrently": "^3.5.0",
    "jasmine-core": "~2.6.2",
    "jasmine-spec-reporter": "~4.1.0",
    "karma": "~1.7.0",
    "karma-chrome-launcher": "~2.1.1",
    "karma-cli": "~1.0.1",
    "karma-coverage-istanbul-reporter": "^1.2.1",
    "karma-jasmine": "~1.1.0",
    "karma-jasmine-html-reporter": "^0.2.2",
    "nodemon": "^1.11.0",
    "protractor": "~5.1.2",
    "tslint": "~5.3.2"
  },
  "engines": {
    "node": "6.11.2"
  }
}
```

Arquivo 6: codigofonte/package.json

```
// Protractor configuration file, see link for more information
// https://github.com/angular/protractor/blob/master/lib/config.ts

const { SpecReporter } = require('jasmine-spec-reporter');

exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './e2e/**/*.e2e-spec.ts'
  ],
  capabilities: {
    'browserName': 'chrome'
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200/',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  onPrepare() {
    require('ts-node').register({
      project: 'e2e/tsconfig.e2e.json'
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: {
        displayStacktrace: true } }));
  }
};
```

Arquivo 7: codigofonte/protractor.conf.js

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

Arquivo 8: codigofonte/proxy.conf.json

```
{
  "folders":
  [
    {
      "folder_exclude_patterns":
      [
        "node_modules",
        "dist"
      ],
      "path": "."
    },
    {
      "path": "/media/feekosta/OneDrive/Onedrive/SISTEMAS DE INFORMAÇÃO/FASE 10/INE5644 - DATA MINING/TRABALHO/Dataset - Dados de intoxicação"
    }
  ]
}
```

Arquivo 9: codigofonte/tcc.sublime–project

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2016",
      "dom"
    ]
  }
}
```

Arquivo 10: codigofonte/tsconfig.json

```
{
  "rulesDirectory": [
    "node_modules/codelyzer"
  ],
  "rules": {
    "arrow-return-shorthand": true,
    "callable-types": true,
    "class-name": true,
    "comment-format": [
      true,
      "check-space"
    ],
    "curly": true,
    "eofline": true,
    "forin": true,
    "import-blacklist": [
      true,
      "rxjs"
    ],
    "import-spacing": true,
    "indent": [
      true,
      "spaces"
    ],
    "interface-over-type-literal": true,
    "label-position": true,
    "max-line-length": [
      true,
      140
    ],
    "member-access": false,
    "member-ordering": [
      true,
      {
        "order": [
          "static-field",
          "instance-field",
          "static-method",
          "instance-method"
        ]
      }
    ],
    "no-arg": true,
    "no-bitwise": true,
    "no-console": [
      true,
      "debug",
      "info",
      "time",
      "timeEnd",
```

```
      "trace"
    ],
    "no-construct": true,
    "no-debugger": true,
    "no-duplicate-super": true,
    "no-empty": false,
    "no-empty-interface": true,
    "no-eval": true,
    "no-inferrable-types": [
      true,
      "ignore-params"
    ],
    "no-misused-new": true,
    "no-non-null-assertion": true,
    "no-shadowed-variable": true,
    "no-string-literal": false,
    "no-string-throw": true,
    "no-switch-case-fall-through": true,
    "no-trailing-whitespace": true,
    "no-unnecessary-initializer": true,
    "no-unused-expression": true,
    "no-use-before-declare": true,
    "no-var-keyword": true,
    "object-literal-sort-keys": false,
    "one-line": [
      true,
      "check-open-brace",
      "check-catch",
      "check-else",
      "check-whitespace"
    ],
    "prefer-const": true,
    "quotemark": [
      true,
      "single"
    ],
    "radix": true,
    "semicolon": [
      true,
      "always"
    ],
    "triple-equals": [
      true,
      "allow-null-check"
    ],
    "typedef-whitespace": [
      true,
      {
        "call-signature": "nospace",
        "index-signature": "nospace",
        "parameter": "nospace",
        "property-declaration": "nospace",
        "variable-declaration": "nospace"
      }
```

```
    ],
    "typeof-compare": true,
    "unified-signatures": true,
    "variable-name": false,
    "whitespace": [
      true,
      "check-branch",
      "check-decl",
      "check-operator",
      "check-separator",
      "check-type"
    ],
    "directive-selector": [
      true,
      "attribute",
      "app",
      "camelCase"
    ],
    "component-selector": [
      true,
      "element",
      "app",
      "kebab-case"
    ],
    "use-input-property-decorator": true,
    "use-output-property-decorator": true,
    "use-host-property-decorator": true,
    "no-input-rename": true,
    "no-output-rename": true,
    "use-life-cycle-interface": true,
    "use-pipe-transform-interface": true,
    "component-class-suffix": true,
    "directive-class-suffix": true,
    "no-access-missing-member": true,
    "templates-use-public": true,
    "invoke-injectable": true
  }
}
```

Arquivo 11: codigofonte/tslint.json

```
.mat-mini-fab{
  margin: 5px;
}

.mat-cell{
  margin-left: 10px;
    margin-right: 10px;
}

.mat-header-cell{
  margin-left: 10px;
    margin-right: 10px;
}
```

Arquivo 12: codigofonte/client/app/_components/alert/alert.component.css

```
<mat-table #table [dataSource]="dataSource">
  <ng-container matColumnDef="status">
    <mat-header-cell *matHeaderCellDef> Status </mat-header-cell>
    <mat-cell *matCellDef="let element">
      <mat-icon mat-list-icon [color]="getStatusColor(element.status)"
          matTooltip="{{getTypeTooltip(element.type)}}"
          matTooltipPosition="right">{{getStatusIcon(element.status)}}
          </mat-icon>
    </mat-cell>
  </ng-container>
  <ng-container matColumnDef="dbUri">
    <mat-header-cell *matHeaderCellDef> URL </mat-header-cell>
    <mat-cell *matCellDef="let element"> {{element.dbUri}} </mat-cell>
  </ng-container>
  <ng-container matColumnDef="collectionName">
    <mat-header-cell *matHeaderCellDef> Coleção </mat-header-cell>
    <mat-cell *matCellDef="let element"> {{element.collectionName}} </
        mat-cell>
  </ng-container>
  <ng-container matColumnDef="date">
    <mat-header-cell *matHeaderCellDef> Data </mat-header-cell>
    <mat-cell *matCellDef="let element"> {{element.date|amCalendar}} <
        /mat-cell>
  </ng-container>
  <ng-container matColumnDef="actions">
    <mat-header-cell *matHeaderCellDef> Ações </mat-header-cell>
    <mat-cell *matCellDef="let element">
      <button mat-mini-fab color="primary" [routerLink]="['/batch',
          element.batchId]" *ngIf="element.status == 'DONE'">
        <mat-icon matTooltip="Visualizar resultado">visibility</
            mat-icon>
      </button>
      <button mat-mini-fab color="warn" (click)="delete(element._id)">
        <mat-icon matTooltip="Excluir alerta">delete</mat-icon>
      </button>
    </mat-cell>
  </ng-container>
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row
      >
  <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
</mat-table>
```

Arquivo 13: codigofonte/client/app/_components/alert/alert.component.html

```
import { async, ComponentFixture, TestBed } from '@angular/core/
    testing';

import { AlertComponent } from './alert.component';

describe('AlertComponent', () => {
```

```
6     let component: AlertComponent;
7     let fixture: ComponentFixture<AlertComponent>;
8
9     beforeEach(async(() => {
10      TestBed.configureTestingModule({
11        declarations: [ AlertComponent ]
12      })
13      .compileComponents();
14    }));
15
16    beforeEach(() => {
17      fixture = TestBed.createComponent(AlertComponent);
18      component = fixture.componentInstance;
19      fixture.detectChanges();
20    });
21
22    it('should create', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 14: codigofonte/client/app/_components/alert/alert.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { DataSource } from '@angular/cdk/table';
3   import { Observable } from 'rxjs/Observable';
4   import { BehaviorSubject } from "rxjs/BehaviorSubject";
5   import { AlertService, FeedbackService } from '../../_services/
        services';
6
7   @Component({
8     selector: 'app-alert',
9     templateUrl: './alert.component.html',
10    styleUrls: ['./alert.component.css']
11  })
12  export class AlertComponent implements OnInit {
13
14    dataSource: AlertSource | null;
15      dataSubject = new BehaviorSubject<any[]>([]);
16      displayedColumns = ['status', 'dbUri', 'collectionName', 'date', '
            actions'];
17
18    constructor(private alertService:AlertService, private
          feedbackService:FeedbackService) { }
19
20    ngOnInit() {
21      this.loadAlerts();
22    }
23
24    loadAlerts(){
25      this.alertService.listAlerts().subscribe({
26        next: value => this.dataSubject.next(value)
27      });
28      if(this.dataSubject){
```

```
29        this.dataSource = new AlertSource(this.dataSubject);
30      }
31    }
32
33    getStatusIcon(alertStatus) {
34      switch(alertStatus){
35        case "ERROR":
36          return "report_problem";
37        case "DONE":
38          return "check_circle";
39      }
40    }
41
42    getStatusColor(alertStatus) {
43      switch(alertStatus){
44        case "ERROR":
45          return "warn";
46        case "DONE":
47          return "primary";
48      }
49    }
50
51    getTypeTooltip(alertType) {
52      switch(alertType){
53        case "DONE":
54          return "Finalizado com sucesso";
55        case "DATABASE_CONNECTION_ERROR":
56          return "Não foi possível conectar ao banco de dados. Verifique
                o endereço informado e tente novamente.";
57        case "EMPTY_COLLECTION_ERROR":
58          return "Não há documentos na coleção informada. Verifique a
              coleção informada e tente novamente.";
59        case "LOADING_DOCUMENTS_ERROR":
60          return "Houve um problema durante a leitura dos documentos.
              Tente novamente.";
61        default:
62          return "Não foi possível concluir a extração. Tente novamente.
              ";
63      }
64    }
65
66    delete(alertId){
67      this.alertService.deleteAlert(alertId).subscribe((data) => {
68        this.feedbackService.success("Deletado");
69        this.loadAlerts();
70      });
71    }
72
73  }
74
75  export class AlertSource extends DataSource<any[]> {
76
77    constructor(private subject: BehaviorSubject<any[]>) {
78      super ();
```

```
  }

  connect (): Observable<any[]> {
    return this.subject.asObservable();
  }

  disconnect (  ): void {
  }

}
```

Arquivo 15: codigofonte/client/app/_components/alert/alert.component.ts

```
.mat-card {
    width: 450px;
  position:absolute;
  top:50%;
  left:50%;
  -ms-transform: translateX(-50%) translateY(-50%);
  -webkit-transform: translate(-50%,-50%);
  transform: translate(-50%,-50%);
}

.app-card-header-title{
  display: inline-flex;
}

.app-card-header-title > .material-icons{
  height: 50px;
    width: 50px;
    font-size: 50px;
}

.app-card-header-title > .mat-card-title{
  font-size: 40px;
  margin-bottom: 0px;
  width: 306px;
    text-align: center;
}

.mat-mini-fab{
  margin: 5px;
}
```

Arquivo 16: codigofonte/client/app/_components/batch/batch.component.css

```
<mat-card>
  <mat-card-header>
    <div class="app-card-header-title">
      <mat-icon mat-list-icon color="primary">description</mat-icon>
        <mat-card-title>{{batch?.collectionName}}</mat-card-title>
      </div>
    </mat-card-header>
```

```
8       <mat-card-content>
9         <mat-list>
10          <mat-list-item>
11            <mat-icon mat-list-icon color="primary">cloud</mat-icon>
12          <h4 mat-line>Url</h4>
13          <p mat-line>{{batch?.dbUri}}</p>
14        </mat-list-item>
15        <mat-list-item>
16            <mat-icon mat-list-icon color="primary">info</mat-icon>
17          <h4 mat-line>Quantidade de registros na coleção</h4>
18          <p mat-line>{{batch?.collectionCount}}</p>
19        </mat-list-item>
20
21        <mat-list-item>
22            <mat-icon mat-list-icon color="primary">info</mat-icon>
23          <h4 mat-line>Registros com mesma estrutura original</h4>
24          <p mat-line>{{batch?.uniqueUnorderedCount}}</p>
25        </mat-list-item>
26        <mat-list-item>
27            <mat-icon mat-list-icon color="primary">info</mat-icon>
28          <h4 mat-line>Registros com mesma estrutura ordenada</h4>
29          <p mat-line>{{batch?.uniqueOrderedCount}}</p>
30        </mat-list-item>
31        <mat-list-item>
32            <mat-icon mat-list-icon color="primary">description</
                mat-icon>
33          <h4 mat-line>JSONSchema</h4>
34          <button mat-mini-fab color="primary" (click)="
                openElapsedTimeModal()">
35            <mat-icon matTooltip="Informações sobre tempo decorrido">
                access_time</mat-icon>
36          </button>
37          <button mat-mini-fab color="primary" [routerLink]="['/
                jsonschema', batch?._id]">
38            <mat-icon matTooltip="Visualizar JSON Schema">visibility</
                mat-icon>
39          </button>
40          <button mat-mini-fab color="accent" (click)="download()">
41            <mat-icon matTooltip="Baixar JSON Schema">get_app</mat-icon>
42          </button>
43        </mat-list-item>
44      </mat-list>
45      </mat-card-content>
46  </mat-card>
```

Arquivo 17: codigofonte/client/app/_components/batch/batch.component.html

```
1   import { async, ComponentFixture, TestBed } from '@angular/core/
        testing';
2
3   import { BatchComponent } from './batch.component';
4
5   describe('BatchComponent', () => {
6     let component: BatchComponent;
```

```
  let fixture: ComponentFixture<BatchComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ BatchComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(BatchComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Arquivo 18: codigofonte/client/app/_components/batch/batch.component.spec.ts

```
import * as FileSaver from 'file-saver';
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { MatDialog } from '@angular/material';
import { BatchElapsedTimeModalComponent } from "../batch-elapsed-time-
    modal/batch-elapsed-time-modal.component";
import { JsonSchemaService } from '../../_services/services';

@Component({
  selector: 'app-batch',
  templateUrl: './batch.component.html',
  styleUrls: ['./batch.component.css']
})
export class BatchComponent implements OnInit {

  batch: any;

  constructor(
    private dialog:MatDialog,
    private route: ActivatedRoute,
    private jsonSchemaService:JsonSchemaService) { }

  ngOnInit() {
    this.route.params.subscribe((params) => {
      this.jsonSchemaService.getBatchById(params['id']).subscribe((
          data) => {
        this.batch = data;
      });
    });
  }

  download() {
```

```
31      this.jsonSchemaService.getJsonSchemaByBatchId(this.batch._id).
            subscribe((data) => {
32        if(data && data.length > 0){
33          const jsonSchema = data[0].jsonSchema;
34          const fileName = `${this.batch.collectionName}.json`;
35          const fileToSave = new Blob([JSON.stringify(jsonSchema)], {
36            type: 'application/json'
37            });
38          FileSaver.saveAs(fileToSave, fileName);
39        }
40      });
41    }
42
43    openElapsedTimeModal(){
44      const batchElapsedTimeModal = this.dialog.open(
            BatchElapsedTimeModalComponent, {
45        height: '500px',
46        width: '500px',
47        data: this.batch
48      });
49      batchElapsedTimeModal.afterClosed().subscribe((result) => {});
50    }
51
52  }
```

Arquivo 19: codigofonte/client/app/_components/batch/batch.component.ts

Arquivo 20: codigofonte/client/app/_components/batch–delete–modal/batch–delete–modal.component.css

```
1  <h1 mat-dialog-title>Deletar</h1>
2  <div mat-dialog-content>
3    <p>Deseja realmente deletar?</p>
4  </div>
5  <div mat-dialog-actions>
6    <button mat-button (click)="onNoClick()" tabindex="-1">Não</button>
7    <button mat-raised-button color="primary" [mat-dialog-close]="'true'
         " tabindex="2">Sim</button>
8  </div>
```

Arquivo 21: codigofonte/client/app/_components/batch–delete–modal/batch–delete–modal.component.htm

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { BatchDeleteModalComponent } from './batch-delete-
       modal.component';
4
5  describe('BatchDeleteModalComponent', () => {
6    let component: BatchDeleteModalComponent;
7    let fixture: ComponentFixture<BatchDeleteModalComponent>;
8
9    beforeEach(async(() => {
```

```
10      TestBed.configureTestingModule({
11        declarations: [ BatchDeleteModalComponent ]
12      })
13      .compileComponents();
14    }));
15
16    beforeEach(() => {
17      fixture = TestBed.createComponent(BatchDeleteModalComponent);
18      component = fixture.componentInstance;
19      fixture.detectChanges();
20    });
21
22    it('should create', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 22: codigofonte/client/app/_components/batch–delete–modal/batch–delete–modal.component.spe

```
1  import { Component, OnInit, Inject } from '@angular/core';
2  import { MatDialog, MatDialogRef, MAT_DIALOG_DATA } from '@angular/
       material';
3
4  @Component({
5    selector: 'app-batch-delete-modal',
6    templateUrl: './batch-delete-modal.component.html',
7    styleUrls: ['./batch-delete-modal.component.css']
8  })
9  export class BatchDeleteModalComponent implements OnInit {
10
11   constructor(
12     public dialogRef: MatDialogRef<BatchDeleteModalComponent>,
13     @Inject(MAT_DIALOG_DATA) public data: any) { }
14
15   onNoClick(): void {
16     this.dialogRef.close();
17   }
18
19   ngOnInit() {
20   }
21
22 }
```

Arquivo 23: codigofonte/client/app/_components/batch–delete–modal/batch–delete–modal.component.ts

Arquivo 24: codigofonte/client/app/_components/batch–elapsed–time–modal/batch–elapsed–time–modal.c

```
1  <h1 mat-dialog-title>Informações detalhadas sobre tempo decorrido em
       cada uma das etapas de extração do JSON Schema</h1>
2  <div mat-dialog-content>
3    <ul>
4        <li>
```

```
5           Etapa 1: Carregar a estrutura dos registros (Raw Schema): <br>
                {{stepOneElapsedTime}}
6         </li>
7         <li>
8           Etapa 2.1: Reduzir os documentos da etapa 1 com mesma
                estrutura (Raw Schema): <br>{{stepTwoDotOneElapsedTime}}
9         </li>
10        <li>
11          Etapa 2.2: Ordenar alfabéticamente a estrutura dos documentos
                da etapa 2.1 e reduzir: <br>{{stepTwoDotTwoElapsedTime}}
12        </li>
13        <li>
14          Etapa 3: Unir a estrutura dos documentos da etapa 2.2: <br> {{
                stepThreeElapsedTime}}
15        </li>
16        <li>
17          Etapa 4: Mapear estrutura do documento resultante da etapa 3
                para JSON Schema: <br>{{stepFourElapsedTime}}
18        </li>
19        <li>
20          Tempo total: <br> {{allStepsElapsedTime}}
21        </li>
22      </ul>
23  </div>
```

Arquivo 25: codigofonte/client/app/_components/batch–elapsed–time–modal/batch–elapsed–time–modal.c

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { BatchElapsedTimeModalComponent } from './batch-elapsed-time-
       modal.component';
4
5  describe('BatchElapsedTimeModalComponent', () => {
6    let component: BatchElapsedTimeModalComponent;
7    let fixture: ComponentFixture<BatchElapsedTimeModalComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ BatchElapsedTimeModalComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(BatchElapsedTimeModalComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', () => {
23     expect(component).toBeTruthy();
24   });
25 });
```

Arquivo 26: codigofonte/client/app/_components/batch-elapsed-time-modal/batch-elapsed-time-modal.c

```
1  import { Component, OnInit, Inject } from '@angular/core';
2  import { MatDialog, MatDialogRef, MAT_DIALOG_DATA } from '@angular/
       material';
3  import * as moment from 'moment';
4
5  @Component({
6    selector: 'app-batch-elapsed-time-modal',
7    templateUrl: './batch-elapsed-time-modal.component.html',
8    styleUrls: ['./batch-elapsed-time-modal.component.css']
9  })
10 export class BatchElapsedTimeModalComponent implements OnInit {
11
12   stepOneElapsedTime: string;
13   stepTwoDotOneElapsedTime: string;
14   stepTwoDotTwoElapsedTime: string;
15   stepThreeElapsedTime: string;
16   stepFourElapsedTime: string;
17   allStepsElapsedTime: string;
18
19   constructor(
20     public dialogRef: MatDialogRef<BatchElapsedTimeModalComponent>,
21     @Inject(MAT_DIALOG_DATA) public batch: any) {
22   }
23
24   ngOnInit() {
25     this.stepOneElapsedTime = this.getTimeDiff(this.batch.startDate,
           this.batch.extractionDate);
26
27     switch (this.batch.reduceType) {
28       case "MAP_REDUCE":
29         this.stepTwoDotOneElapsedTime = this.getTimeDiff(
               this.batch.extractionDate,
               this.batch.unorderedMapReduceDate);
30         this.stepTwoDotTwoElapsedTime = this.getTimeDiff(
               this.batch.unorderedMapReduceDate,
               this.batch.orderedMapReduceDate);
31         this.stepThreeElapsedTime = this.getTimeDiff(
               this.batch.orderedMapReduceDate, this.batch.unionDate);
32         break;
33
34       case "AGGREGATE":
35         this.stepTwoDotOneElapsedTime = this.getTimeDiff(
               this.batch.extractionDate,
               this.batch.unorderedAggregationDate);
36         this.stepTwoDotTwoElapsedTime = this.getTimeDiff(
               this.batch.unorderedAggregationDate,
               this.batch.orderedAggregationDate);
37         this.stepThreeElapsedTime = this.getTimeDiff(
               this.batch.orderedAggregationDate, this.batch.unionDate);
38         break;
```

```
39
40        case "AGGREGATE_AND_MAP_REDUCE":
41          this.stepTwoDotOneElapsedTime = this.getTimeDiff(
                this.batch.extractionDate,
                this.batch.unorderedAggregationDate);
42          this.stepTwoDotTwoElapsedTime = this.getTimeDiff(
                this.batch.unorderedAggregationDate,
                this.batch.orderedMapReduceDate);
43          this.stepThreeElapsedTime = this.getTimeDiff(
                this.batch.orderedMapReduceDate, this.batch.unionDate);
44          break;
45      }
46
47      this.stepFourElapsedTime = this.getTimeDiff(this.batch.unionDate,
            this.batch.endDate);
48      this.allStepsElapsedTime = this.getTimeDiff(this.batch.startDate,
            this.batch.endDate);
49    }
50
51    private getTimeDiff(date1: Date, date2: Date){
52      let ms = moment(date2).diff(moment(date1));
53      let d = moment.duration(ms);
54      return Math.floor(d.asHours()) + moment.utc(ms).format(":mm:ss:SSS
            ");
55    }
56
57    onNoClick(): void {
58      this.dialogRef.close();
59    }
60
61  }
```

Arquivo 27: codigofonte/client/app/_components/batch–elapsed–time–modal/batch–elapsed–time–modal.c

```
1  export * from './alert/alert.component';
2  export * from './discovery/discovery.component';
3  export * from './feedback/feedback.component';
4  export * from './home/home.component';
5  export * from './login/login.component';
6  export * from './navbar/navbar.component';
7  export * from './register/register.component';
8  export * from './schemes/schemes.component';
9  export * from './user/user.component';
10 export * from './batch/batch.component';
11 export * from './batch-elapsed-time-modal/batch-elapsed-time-
       modal.component';
12 export * from './batch-delete-modal/batch-delete-modal.component';
13 export * from './json-schema/json-schema.component';
14 export * from './loading/loading.component';
```

Arquivo 28: codigofonte/client/app/_components/components.ts

```
1  .mat-form-field {
```

```
2     display: inline;
3   }
4
5   .mat-stepper-horizontal {
6     width: 500px;
7       margin-left: auto;
8       margin-right: auto;
9   }
10
11  .mat-select {
12    padding-bottom: 17.5px;
13  }
```

Arquivo 29: codigofonte/client/app/_components/discovery/discovery.component.css

```
1   <form name="form" #f="ngForm">
2     <mat-horizontal-stepper [linear]="isLinear">
3
4       <mat-step>
5         <ng-template matStepLabel>Conexão</ng-template>
6         <mat-form-field>
7           <input matInput placeholder="Endereço" [(ngModel)]="
                model.address" name="address" required>
8         </mat-form-field>
9         <mat-form-field>
10          <input matInput placeholder="Porta" [(ngModel)]="model.port"
                name="port" required>
11        </mat-form-field>
12        <mat-form-field>
13          <input matInput placeholder="Nome do banco" [(ngModel)]="
                model.databaseName" name="databaseName" required>
14        </mat-form-field>
15        <mat-form-field>
16          <input matInput placeholder="Nome da coleção" [(ngModel)]="
                model.collectionName" name="collectionName" required>
17        </mat-form-field>
18        <div>
19          <button mat-button [routerLink]="['/']">Cancelar</button>
20          <button mat-raised-button color="primary" matStepperNext [
                disabled]="f.form.invalid">Próximo</button>
21        </div>
22      </mat-step>
23
24      <mat-step>
25        <ng-template matStepLabel>Autenticação</ng-template>
26        <mat-form-field>
27          <input matInput placeholder="Nome do banco de dados de
                administração" [(ngModel)]="
                model.authentication.authDatabase" name="authDatabase">
28        </mat-form-field>
29        <mat-form-field>
30          <input matInput placeholder="Usuário" [(ngModel)]="
                model.authentication.userName" name="userName">
31        </mat-form-field>
```

```
32        <mat-form-field>
33          <input matInput type="password" placeholder="Senha" [(ngModel)
              ]="model.authentication.password" name="password">
34        </mat-form-field>
35        <mat-select placeholder="Mecanismo de autenticação" [(ngModel)]=
              "model.authentication.authMechanism" name="authMechanism">
36          <mat-option *ngFor="let authMechanism of authMechanisms" [
              value]="authMechanism">{{authMechanism}}</mat-option>
37        </mat-select>
38        <div>
39          <button mat-button matStepperPrevious>Voltar</button>
40          <button mat-raised-button color="primary" matStepperNext [
              disabled]="f.form.invalid">Próximo</button>
41        </div>
42      </mat-step>
43
44      <mat-step>
45        <ng-template matStepLabel>Pronto</ng-template>
46        <p>
47        Esse processo pode demorar de acordo com o tamanho da coleção de
              documentos informado.
48        Você receberá uma notificação assim que o JSONSchema for extraí
              do. </p>
49        <div>
50          <button mat-button matStepperPrevious>Voltar</button>
51          <button mat-raised-button color="primary" [disabled]="
              f.form.invalid" (click)="f.form.valid && discovery()">
              Extrair JSONSchema</button>
52        </div>
53      </mat-step>
54    </mat-horizontal-stepper>
55  </form>
```

Arquivo 30: codigofonte/client/app/_components/discovery/discovery.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { DiscoveryComponent } from './discovery.component';
4
5  describe('DiscoveryComponent', () => {
6    let component: DiscoveryComponent;
7    let fixture: ComponentFixture<DiscoveryComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ DiscoveryComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(DiscoveryComponent);
18     component = fixture.componentInstance;
```

```
19      fixture.detectChanges();
20    });
21
22    it('should be created', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 31: codigofonte/client/app/_components/discovery/discovery.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { Router } from '@angular/router';
3   import { FormBuilder, FormGroup, Validators } from '@angular/forms';
4   import { DatabaseParam } from '../../_params/params';
5   import { JsonSchemaService, FeedbackService } from '../../_services/
        services';
6
7   @Component({
8     selector: 'app-discovery',
9     templateUrl: './discovery.component.html',
10    styleUrls: ['./discovery.component.css']
11  })
12  export class DiscoveryComponent implements OnInit {
13
14    loading: boolean = false;
15    isLinear: boolean = false;
16      formGroup: FormGroup;
17      model: DatabaseParam;
18      authMechanisms: Array<string>;
19
20      constructor(
21        private _formBuilder: FormBuilder,
22      private router: Router,
23      private jsonSchemaService: JsonSchemaService,
24      private feedbackService: FeedbackService) {
25        this.model = new DatabaseParam();
26      this.model.port = "27017";
27      this.authMechanisms = [
28        "DEFAULT", "GSSAPI", "PLAIN", "MONGODB-X509", "SCRAM-SHA-1", "
              MONGODB-CR"
29      ]
30      }
31
32    ngOnInit() {
33      this.formGroup = this._formBuilder.group({
34        formCtrl: ['', Validators.required]
35      });
36    }
37
38    discovery(){
39      this.loading = true;
40      this.jsonSchemaService.discovery(this.model)
41        .subscribe(
42          data => {
```

```
43            this.feedbackService.success("Processo iniciado", true);
44            this.router.navigate(["/"]);
45          },
46          error => {
47            this.feedbackService.error(error.json().error);
48            this.loading = false;
49          }
50        );
51
52    }
53
54  }
```

Arquivo 32: codigofonte/client/app/_components/discovery/discovery.component.ts

Arquivo 33: codigofonte/client/app/_components/feedback/feedback.component.css

```
1  <div *ngIf="message" [ngClass]="{ 'alert': message, 'alert-success':
       message.type === 'success', 'alert-danger': message.type === '
       error' }">{{message.text}}</div>
```

Arquivo 34: codigofonte/client/app/_components/feedback/feedback.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { FeedbackComponent } from './feedback.component';
4
5  describe('FeedbackComponent', () => {
6    let component: FeedbackComponent;
7    let fixture: ComponentFixture<FeedbackComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ FeedbackComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(FeedbackComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', () => {
23     expect(component).toBeTruthy();
24   });
25 });
```

Arquivo 35: codigofonte/client/app/_components/feedback/feedback.component.spec.ts

```
import { Component, OnInit } from '@angular/core';
import { FeedbackService } from '../../_services/services';

@Component({
  selector: 'app-feedback',
  templateUrl: './feedback.component.html',
  styleUrls: ['./feedback.component.css']
})
export class FeedbackComponent implements OnInit {

  message: any;

  constructor(private feedbackService: FeedbackService) { }

  ngOnInit() {
    this.feedbackService.getMessage().subscribe(message => {
        this.message = message; });
  }

}
```

Arquivo 36: codigofonte/client/app/_components/feedback/feedback.component.ts

```
.app-action {
  display: inline-block;
  position: fixed;
    bottom: 20px;
    right: 20px;
}

.app-mat-spinner{
  height: 32px;
    width: 32px;
}

.app-mat-spinner svg{
  height: 100% !important;
    width: 100% !important;
}

.mat-mini-fab{
  margin: 5px;
}

.mat-list{
  margin-right: 80px;
}
```

Arquivo 37: codigofonte/client/app/_components/home/home.component.css

```
<mat-list>
  <mat-list-item *ngFor="let batch of batches">
    <h4 mat-line>{{batch.dbUri}}</h4>
```

```
4       <h6 mat-line>{{batch.collectionName}}</h6>
5       <mat-icon matTooltip="{{getTypeTooltip(batch.statusType)}}"
            matTooltipPosition="right" [color]="getStatusColor(
            batch.status)" mat-list-icon *ngIf="batch.status == 'DONE'||
            batch.status == 'ERROR'">{{getStatusIcon(batch.status)}}</
            mat-icon>
6     <mat-icon matTooltip="Em andamento" matTooltipPosition="right"
          mat-list-icon *ngIf="batch.status != 'DONE'&&batch.status != '
          ERROR'">
7       <mat-spinner class="app-mat-spinner"></mat-spinner>
8     </mat-icon>
9     <button matTooltip="Visualizar" mat-mini-fab color="primary" [
          routerLink]="['/batch', batch._id]" *ngIf="batch.status == 'DONE
          '">
10      <mat-icon>visibility</mat-icon>
11    </button>
12    <button matTooltip="Deletar" mat-mini-fab color="warn" (click)="
          delete(batch._id)" *ngIf="batch.status == 'DONE'||batch.status
          == 'ERROR'">
13      <mat-icon>delete</mat-icon>
14    </button>
15    </mat-list-item>
16  </mat-list>
17  <span class="app-action">
18    <button mat-fab matTooltip="Executar nova extração de JSON Schema"
          matTooltipPosition="left" [routerLink]="['/discovery']">
          <mat-icon>add</mat-icon></button>
19  </span>
```

Arquivo 38: codigofonte/client/app/_components/home/home.component.html

```
1   import { async, ComponentFixture, TestBed } from '@angular/core/
        testing';
2
3   import { HomeComponent } from './home.component';
4
5   describe('HomeComponent', () => {
6     let component: HomeComponent;
7     let fixture: ComponentFixture<HomeComponent>;
8
9     beforeEach(async(() => {
10      TestBed.configureTestingModule({
11        declarations: [ HomeComponent ]
12      })
13      .compileComponents();
14    }));
15
16    beforeEach(() => {
17      fixture = TestBed.createComponent(HomeComponent);
18      component = fixture.componentInstance;
19      fixture.detectChanges();
20    });
21
22    it('should be created', () => {
```

```
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 39: codigofonte/client/app/_components/home/home.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { MatDialog } from '@angular/material';
3   import { User } from '../../_models/user';
4   import { JsonSchemaService, FeedbackService } from "../../_services/
        services";
5   import { BatchDeleteModalComponent } from "../batch-delete-modal/batch
        -delete-modal.component";
6
7   @Component({
8     selector: 'app-home',
9     templateUrl: './home.component.html',
10    styleUrls: ['./home.component.css']
11  })
12  export class HomeComponent implements OnInit {
13
14    currentUser: User;
15    batches: any;
16
17    constructor(
18      private dialog:MatDialog,
19      private jsonSchemaService:JsonSchemaService,
20      private feedbackService: FeedbackService) { }
21
22    ngOnInit() {
23      this.currentUser = JSON.parse(localStorage.getItem('currentUser'))
            ;
24      this.jsonSchemaService.listBatches().subscribe((data) => {
25            this.batches = data;
26          });
27    }
28
29    delete(batchId) {
30      const deleteModal = this.dialog.open(BatchDeleteModalComponent, {
31        width: '250px',
32        data: { 'batchId': batchId }
33      });
34      deleteModal.afterClosed().subscribe((result) => {
35        if(result){
36          this.jsonSchemaService.deleteBatch(batchId).subscribe((data)
                => {
37            this.feedbackService.success("Deletado");
38            this.jsonSchemaService.listBatches().subscribe((data) => {
39                  this.batches = data;
40                });
41          });
42        }
43      });
44    }
```

```
45
46    getStatusIcon(batchStatus) {
47      switch(batchStatus){
48        case "ERROR":
49          return "report_problem";
50        case "DONE":
51          return "check_circle";
52      }
53    }
54
55    getStatusColor(batchStatus) {
56      switch(batchStatus){
57        case "ERROR":
58          return "warn";
59        default:
60          return "primary";
61      }
62    }
63
64    getTypeTooltip(alertType) {
65      switch(alertType){
66        case "DONE":
67          return "Finalizado com sucesso";
68        case "DATABASE_CONNECTION_ERROR":
69          return "Não foi possível conectar ao banco de dados. Verifique
               o endereço informado e tente novamente.";
70        case "EMPTY_COLLECTION_ERROR":
71          return "Não há documentos na coleção informada. Verifique a
              mesma e tente novamente.";
72        case "LOADING_DOCUMENTS_ERROR":
73          return "Houve um problema durante a leitura dos documentos.
              Tente novamente.";
74        default:
75          return "Não foi possível concluir a extração. Tente novamente.
              ";
76      }
77    }
78
79
80  }
```

Arquivo 40: codigofonte/client/app/_components/home/home.component.ts

Arquivo 41: codigofonte/client/app/_components/json–schema/json–schema.component.css

```
1  <pre [innerHTML]="jsonSchema | prettyJson"></pre>
```

Arquivo 42: codigofonte/client/app/_components/json–schema/json–schema.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
```

```
3  import { JsonSchemaComponent } from './json-schema.component';
4
5  describe('JsonSchemaComponent', () => {
6    let component: JsonSchemaComponent;
7    let fixture: ComponentFixture<JsonSchemaComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ JsonSchemaComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(JsonSchemaComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', () => {
23     expect(component).toBeTruthy();
24   });
25 });
```

Arquivo 43: codigofonte/client/app/_components/json–schema/json–schema.component.spec.ts

```
1  import { Component, OnInit } from '@angular/core';
2  import { ActivatedRoute } from '@angular/router';
3  import { JsonSchemaService } from '../../_services/services';
4
5  @Component({
6    selector: 'app-json-schema',
7    templateUrl: './json-schema.component.html',
8    styleUrls: ['./json-schema.component.css']
9  })
10 export class JsonSchemaComponent implements OnInit {
11
12   jsonSchema: any;
13
14   constructor(
15   private route: ActivatedRoute,
16   private jsonSchemaService:JsonSchemaService) { }
17
18
19   ngOnInit() {
20     this.route.params.subscribe((params) => {
21       this.jsonSchemaService.getJsonSchemaByBatchId(params['id']).
             subscribe((data) => {
22         if(data && data.length > 0){
23           this.jsonSchema = data[0].jsonSchema;
24         }
25       });
26     });
27   }
```

```

}
```

Arquivo 44: codigofonte/client/app/_components/json–schema/json–schema.component.ts

Arquivo 45: codigofonte/client/app/_components/loading/loading.component.css

```
<mat-progress-bar color="primary" mode="indeterminate" *ngIf="visible"
    ></mat-progress-bar>
```

Arquivo 46: codigofonte/client/app/_components/loading/loading.component.html

```
import { async, ComponentFixture, TestBed } from '@angular/core/
    testing';

import { LoadingComponent } from './loading.component';

describe('LoadingComponent', () => {
  let component: LoadingComponent;
  let fixture: ComponentFixture<LoadingComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ LoadingComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(LoadingComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Arquivo 47: codigofonte/client/app/_components/loading/loading.component.spec.ts

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { Subscription } from 'rxjs/Rx';
import { LoadingState, LoadingService } from '../../_services/loading/
    loading.service';

@Component({
  selector: 'app-loading',
  templateUrl: './loading.component.html',
  styleUrls: ['./loading.component.css']
})
export class LoadingComponent implements OnDestroy, OnInit {
```

```

  public visible: boolean = false;
  private _loadingStateChanged: Subscription;

  constructor(private _loadingService: LoadingService) { }

  ngOnInit() {
    this._loadingStateChanged = this._loadingService.loadingState
      .subscribe((state: LoadingState) => this.visible = state.show);
  }

  ngOnDestroy() {
    this._loadingStateChanged.unsubscribe();
  }

}
```

Arquivo 48: codigofonte/client/app/_components/loading/loading.component.ts

```
.mat-card {
    width: 400px;
    padding: 30px;
  position:absolute;
  top:50%;
  left:50%;
  -ms-transform: translateX(-50%) translateY(-50%);
  -webkit-transform: translate(-50%,-50%);
  transform: translate(-50%,-50%);
}

.mat-card-actions {
  margin-left: 0px;
    margin-right: 0px;
    padding: 0px 0;
}

.mat-form-field {
  display: inline;
}

.app-logo {
  display: block;
    margin-left: auto;
    margin-right: auto;
    width: 250px;
    height: 90px;
    margin-bottom: 20px;
    margin-top: 20px !important;
}
```

Arquivo 49: codigofonte/client/app/_components/login/login.component.css

```
<mat-card>
```

```
2
3     <img src="../../../assets/jsd_logo.png" alt="JSON Schema Discovery"
          class="app-logo">
4
5     <form name="form" (ngSubmit)="f.form.valid && login()" #f="ngForm"
          novalidate>
6
7       <mat-card-content>
8
9         <mat-input-container class="mat-block" >
10          <input matInput type="email" placeholder="Email" required [
                formControl]="emailFormControl" [(ngModel)]="model.email"
                />
11          <mat-error *ngIf="emailFormControl.hasError('pattern')">
12            Por favor, digite um email válido
13          </mat-error>
14          <mat-error *ngIf="emailFormControl.hasError('required')">
15            Email é <strong>obrigatório</strong>
16          </mat-error>
17        </mat-input-container>
18
19        <mat-input-container class="mat-block">
20          <input matInput placeholder="Senha" [type]="showPassword ? '
                text' : 'password'" required [(ngModel)]="model.password"
                #password="ngModel" name="password">
21          <mat-icon matSuffix (click)="showPassword = !showPassword">{{
                showPassword ? 'visibility' : 'visibility_off'}}</mat-icon
                >
22          <mat-error *ngIf="!password.valid">
23            Senha é <strong>obrigatório</strong>
24          </mat-error>
25        </mat-input-container>
26
27      </mat-card-content>
28
29      <mat-card-actions>
30        <button mat-raised-button color="primary" [disabled]="loading">
              Entrar</button>
31        <a mat-raised-button [routerLink]="['/register']">Criar conta</a
              >
32      </mat-card-actions>
33
34    </form>
35
36  </mat-card>
```

Arquivo 50: codigofonte/client/app/_components/login/login.component.html

```
1   import { async, ComponentFixture, TestBed } from '@angular/core/
        testing';
2
3   import { LoginComponent } from './login.component';
4
5   describe('LoginComponent', () => {
```

```
6     let component: LoginComponent;
7     let fixture: ComponentFixture<LoginComponent>;
8
9     beforeEach(async(() => {
10      TestBed.configureTestingModule({
11        declarations: [ LoginComponent ]
12      })
13      .compileComponents();
14    }));
15
16    beforeEach(() => {
17      fixture = TestBed.createComponent(LoginComponent);
18      component = fixture.componentInstance;
19      fixture.detectChanges();
20    });
21
22    it('should be created', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 51: codigofonte/client/app/_components/login/login.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { ActivatedRoute, Router } from '@angular/router';
3   import { FormControl, Validators } from '@angular/forms';
4   import { AuthenticationService, EventService, FeedbackService } from '
        ../../_services/services';
5
6   const EMAIL_REGEX = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0
        -9-]+(?:\.[a-zA-Z0-9-]+)*$/;
7
8   @Component({
9     selector: 'app-login',
10    templateUrl: './login.component.html',
11    styleUrls: ['./login.component.css']
12  })
13  export class LoginComponent implements OnInit {
14
15    model: any = {};
16      loading: boolean = false;
17      showPassword: boolean = false;
18      returnUrl: string;
19
20    constructor(
21      private route: ActivatedRoute,
22      private router: Router,
23      private authenticationService: AuthenticationService,
24      private eventService: EventService,
25      private feedbackService: FeedbackService) { }
26
27    emailFormControl = new FormControl('', [
28        Validators.required,
29        Validators.pattern(EMAIL_REGEX)
```

```
30    ]);
31
32    ngOnInit() {
33      // reset login status
34      this.authenticationService.logout();
35      // get return url from route parameters or default to '/'
36      this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '
            /';
37    }
38
39    login() {
40      if(this.model.email && this.model.password){
41        this.loading = true;
42        this.authenticationService.login(this.model.email,
              this.model.password)
43          .subscribe(
44            data => {
45              this.eventService.setIsLogged(true);
46              this.router.navigate([this.returnUrl]);
47            },
48            error => {
49              this.feedbackService.error(error.json().error);
50              this.loading = false;
51            }
52          );
53      }
54    }
55
56  }
```

Arquivo 52: codigofonte/client/app/_components/login/login.component.ts

```
1   .app-divider {
2       height: 1px;
3       overflow: hidden;
4       background-color: #e0e0e0;
5       margin-top: 20px;
6       margin-bottom: 20px;
7   }
8
9   .app-logo-side-nav {
10      padding-bottom: 20px;
11      padding-top: 20px;
12  }
13
14  .app-side-nav-button{
15    text-align: left;
16    width: 100%;
17    height: 100%;
18  }
19
20  .button-open-side-nav{
21    line-height: 40px;
22  }
```

```
23
24  .mat-drawer-container {
25    position: fixed;
26    height: 100%;
27    min-height: 100%;
28    width: 100%;
29    min-width: 100%;
30  }
31
32  .mat-drawer {
33    padding: 10px;
34    min-width: 170px;
35  }
36
37  .mat-card {
38    background: rgba(0,0,0,0.0);
39    box-shadow: none !important;
40    color: hsla(0,0%,100%,.87);
41    padding: 14px;
42  }
43
44  .mat-button {
45    min-width: 0px;
46    padding: 0px;
47  }
48
49  .app-toolbar-title {
50    background: #3f51b5;
51    margin-left: 10px;
52    color: hsla(0,0%,100%,.87);
53  }
54
55  .app-toolbar-title:focus,
56  .app-toolbar-title:hover,
57  .app-toolbar-title:active{
58    background: #3f51b5;
59  }
60
61  .app-toolbar-filler {
62    flex: 1 1 auto;
63  }
64
65  .app-toolbar-notification {
66    top: 8px;
67    background: #f36;
68    border-radius: 3px;
69    left: 20px;
70    padding: 0 3px;
71    position: absolute;
72    font-size: 10px;
73    line-height: 13px;
74  }
```

Arquivo 53: codigofonte/client/app/_components/navbar/navbar.component.css

```
<mat-sidenav-container >

  <mat-sidenav mode="side" #sidenav>

    <mat-list>

      <mat-list-item class="app-logo-side-nav">
        <img src="../../../assets/jsd_logo.png" alt="JSON Schema
            Discovery">
      </mat-list-item>

      <mat-list-item class="app-divider"></mat-list-item>

      <mat-list-item>
        <button mat-button color="primary" class="app-side-nav-button"
            [routerLink]="['/']">
          <i class="material-icons">home</i>
          WORKSPACE
        </button>
      </mat-list-item>

      <mat-list-item>
        <button mat-button color="primary" class="app-side-nav-button"
            [routerLink]="['/alert']">
          <i class="material-icons">notifications</i>
          ALERTAS
        </button>
      </mat-list-item>

      <mat-list-item class="app-divider"></mat-list-item>

      <mat-list-item>
        <button mat-button color="primary" class="app-side-nav-button"
            (click)="logout();sidenav.close()">
          <i class="material-icons">exit_to_app</i>
          LOGOUT
        </button>
      </mat-list-item>

    </mat-list>
  </mat-sidenav>

  <mat-toolbar color="primary" *ngIf="isLogged">

    <button class="mat-icon-button" (click)="sidenav.toggle()">
      <i class="material-icons button-open-side-nav">menu</i>
    </button>

    <button class="app-toolbar-title" mat-menu-item [routerLink]="
        ['/']">JSONSchema Discovery</button>
```

```
46
47      <button class="mat-icon-button" mat-button [routerLink]="['/alert
            ']">
48        <i class="material-icons" [hidden]="!alertsCount">notifications<
            /i>
49        <i class="material-icons" [hidden]="alertsCount">
            notifications_none</i>
50        <span class="app-toolbar-notification" [hidden]="!alertsCount">
            {{alertsCount}}</span>
51      </button>
52
53      <button class="mat-icon-button" mat-button [mat-menu-trigger-for]=
            "menu">
54        <i class="material-icons">account_circle</i>
55      </button>
56
57      <mat-menu #menu="matMenu">
58        <button mat-menu-item [routerLink]="['/user']">Perfil</button>
59        <button mat-menu-item (click)="logout();sidenav.close()">Sair
            </button>
60      </mat-menu>
61
62    </mat-toolbar>
63
64    <app-feedback></app-feedback>
65
66    <app-loading></app-loading>
67
68    <router-outlet></router-outlet>
69
70  </mat-sidenav-container>
```

Arquivo 54: codigofonte/client/app/_components/navbar/navbar.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { NavbarComponent } from './navbar.component';
4
5  describe('NavbarComponent', () => {
6    let component: NavbarComponent;
7    let fixture: ComponentFixture<NavbarComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ NavbarComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(NavbarComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
```

```
20    });
21
22    it('should be created', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 55: codigofonte/client/app/_components/navbar/navbar.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { Router } from '@angular/router';
3   import { AuthenticationService, EventService, AlertService } from "
        ../../_services/services";
4
5   @Component({
6     selector: 'app-navbar',
7     templateUrl: './navbar.component.html',
8     styleUrls: ['./navbar.component.css']
9   })
10  export class NavbarComponent implements OnInit {
11
12    isLogged: boolean = false;
13    alertsCount: number;
14
15    constructor(
16      private router: Router,
17      private authenticationService: AuthenticationService,
18      private eventService: EventService,
19      private alertService: AlertService) {
20
21    }
22
23    ngOnInit() {
24      this.eventService.isLoggedEmitter.subscribe((mode)=>{
25          if (mode !== null) {
26            this.isLogged = mode;
27          }
28          if(this.isLogged){
29            this.alertService.countAlerts().subscribe((count)=>{
30              this.alertsCount = count;
31            });
32          }
33      });
34    }
35
36    logout(){
37      this.authenticationService.logout();
38      this.isLogged = false;
39      this.router.navigate(["/"]);
40      this.router.navigate(["/login"]);
41    }
42
43  }
```

Arquivo 56: codigofonte/client/app/_components/navbar/navbar.component.ts

```
.mat-card {
    width: 400px;
    padding: 30px;
  position:absolute;
  top:50%;
  left:50%;
  -ms-transform: translateX(-50%) translateY(-50%);
  -webkit-transform: translate(-50%,-50%);
  transform: translate(-50%,-50%);
}

.mat-card-actions {
  margin-left: 0px;
    margin-right: 0px;
    padding: 0px 0;
}

.mat-form-field {
  display: inline;
}

.app-logo {
  display: block;
    margin-left: auto;
    margin-right: auto;
    width: 250px;
    height: 90px;
    margin-bottom: 20px;
    margin-top: 20px !important;
}
```

Arquivo 57: codigofonte/client/app/_components/register/register.component.css

```
<mat-card>

  <img src="../../../assets/jsd_logo.png" alt="JSON Schema Discovery"
      class="app-logo">

  <form name="form" (ngSubmit)="f.form.valid && register()" #f="ngForm
      " novalidate>

    <mat-card-content>

      <mat-input-container class="mat-block">
        <input matInput placeholder="Nome" [(ngModel)]="model.username
            " #username="ngModel" name="username" required/>
        <mat-error *ngIf="!username.valid">
          Nome é <strong>obrigatório</strong>
        </mat-error>
      </mat-input-container>
```

```
15
16        <mat-input-container class="mat-block">
17          <input type="email" matInput placeholder="Email" [formControl]
                ="emailFormControl" [(ngModel)]="model.email" required/>
18          <mat-error *ngIf="emailFormControl.hasError('pattern')">
19            Por favor, digite um email válido
20          </mat-error>
21          <mat-error *ngIf="emailFormControl.hasError('required')">
22            Email é <strong>obrigatório</strong>
23          </mat-error>
24        </mat-input-container>
25
26        <mat-input-container class="mat-block">
27          <input matInput placeholder="Criar senha" [type]="showPassword
                ? 'text' : 'password'" required [(ngModel)]="
                model.password" #password="ngModel" name="password"
                minlength="6">
28          <mat-icon matSuffix (click)="showPassword = !showPassword">{{
                showPassword ? 'visibility' : 'visibility_off'}}</mat-icon
                >
29          <mat-error *ngIf="!password.valid">
30            Senha é <strong>obrigatório</strong>
31          </mat-error>
32        </mat-input-container>
33
34      </mat-card-content>
35
36      <mat-card-actions>
37        <button mat-raised-button color="primary" [disabled]="loading">
              Cadastrar-me</button>
38        <a mat-raised-button [routerLink]="['/login']">Cancelar</a>
39      </mat-card-actions>
40
41    </form>
42
43  </mat-card>
```

Arquivo 58: codigofonte/client/app/_components/register/register.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { RegisterComponent } from './register.component';
4
5  describe('RegisterComponent', () => {
6    let component: RegisterComponent;
7    let fixture: ComponentFixture<RegisterComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ RegisterComponent ]
12     })
13     .compileComponents();
14   }));
```

```
15
16    beforeEach(() => {
17      fixture = TestBed.createComponent(RegisterComponent);
18      component = fixture.componentInstance;
19      fixture.detectChanges();
20    });
21
22    it('should be created', () => {
23      expect(component).toBeTruthy();
24    });
25  });
```

Arquivo 59: codigofonte/client/app/_components/register/register.component.spec.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormControl, Validators } from '@angular/forms';
3   import { Router } from '@angular/router';
4   import { AuthenticationService, FeedbackService, RegistrationService }
        from '../../_services/services';
5
6   const EMAIL_REGEX = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0
        -9-]+(?:\.[a-zA-Z0-9-]+)*$/;
7
8   @Component({
9     selector: 'app-register',
10    templateUrl: 'register.component.html',
11    styleUrls: ['register.component.css']
12  })
13  export class RegisterComponent implements OnInit {
14
15    model: any = {};
16      loading = false;
17      showPassword: boolean = false;
18
19    constructor(
20      private router: Router,
21      private authenticationService: AuthenticationService,
22      private registrationService: RegistrationService,
23      private feedbackService: FeedbackService) { }
24
25    emailFormControl = new FormControl('', [
26        Validators.required,
27        Validators.pattern(EMAIL_REGEX)
28    ]);
29
30    ngOnInit() {
31    }
32
33    register() {
34      if(this.model.email && this.model.password && this.model.username)
            {
35        this.loading = true;
36        this.registrationService.register(this.model)
37          .subscribe(
```

```
38            data => {
39              this.authenticationService.login(this.model.email,
                    this.model.password)
40                .subscribe(
41                  data => {
42                    this.feedbackService.success('Bem Vindo', true);
43                    this.router.navigate(['/']);
44                  }
45                );
46            },
47            error => {
48              this.feedbackService.error(error.json().error);
49              this.loading = false;
50            }
51          );
52      }
53    }
54
55
56
57  }
```

Arquivo 60: codigofonte/client/app/_components/register/register.component.ts

Arquivo 61: codigofonte/client/app/_components/schemes/schemes.component.css

```
1  <p>
2    schemes works!
3  </p>
```

Arquivo 62: codigofonte/client/app/_components/schemes/schemes.component.html

```
1  import { async, ComponentFixture, TestBed } from '@angular/core/
       testing';
2
3  import { SchemesComponent } from './schemes.component';
4
5  describe('SchemesComponent', () => {
6    let component: SchemesComponent;
7    let fixture: ComponentFixture<SchemesComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ SchemesComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(SchemesComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
```

```
  });

  it('should be created', () => {
    expect(component).toBeTruthy();
  });
});
```

Arquivo 63: codigofonte/client/app/_components/schemes/schemes.component.spec.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-schemes',
  templateUrl: './schemes.component.html',
  styleUrls: ['./schemes.component.css']
})
export class SchemesComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Arquivo 64: codigofonte/client/app/_components/schemes/schemes.component.ts

```
.mat-card {
    width: 400px;
  position:absolute;
  top:50%;
  left:50%;
  -ms-transform: translateX(-50%) translateY(-50%);
  -webkit-transform: translate(-50%,-50%);
  transform: translate(-50%,-50%);
}

.mat-card-actions {
  margin-left: 0px;
    margin-right: 0px;
    padding: 0px 0;
}

.mat-form-field {
  display: inline;
}


.app-user-edit-button{
  position: absolute;
  right: 0px;
  top: 10px;
}
```

Arquivo 65: codigofonte/client/app/_components/user/user.component.css

```
<mat-card >
  <mat-card-content>
    <mat-list>
      <mat-list-item>
        <mat-icon mat-list-icon color="primary">person</mat-icon>
        <h4 mat-line>Nome</h4>
        <p mat-line>{{user?.username}}</p>
      </mat-list-item>
      <mat-list-item>
        <mat-icon mat-list-icon color="primary">email</mat-icon>
        <h4 mat-line>Email</h4>
        <p mat-line>{{user?.email}}</p>
      </mat-list-item>
      <mat-list-item>
        <mat-icon mat-list-icon color="primary">vpn_key</mat-icon>
        <h4 mat-line>Senha</h4>
        <p mat-line><mat-icon matTooltip="Editar senha">more_horiz</
            mat-icon></p>
      </mat-list-item>
    </mat-list>
  </mat-card-content>
</mat-card>
```

Arquivo 66: codigofonte/client/app/_components/user/user.component.html

```
import { async, ComponentFixture, TestBed } from '@angular/core/
    testing';

import { UserComponent } from './user.component';

describe('UserComponent', () => {
  let component: UserComponent;
  let fixture: ComponentFixture<UserComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ UserComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(UserComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
```

```
25 });
```

Arquivo 67: codigofonte/client/app/_components/user/user.component.spec.ts

```
1  import { Component, OnInit } from '@angular/core';
2  import { UserService } from '../../_services/services';
3
4  @Component({
5    selector: 'app-user',
6    templateUrl: './user.component.html',
7    styleUrls: ['./user.component.css']
8  })
9  export class UserComponent implements OnInit {
10
11     user: any = {};
12
13   constructor(private userService: UserService) { }
14
15   ngOnInit() {
16     this.userService.get().subscribe((data) => {
17       if(data){
18         this.user = data;
19       }
20     });
21   }
22
23 }
```

Arquivo 68: codigofonte/client/app/_components/user/user.component.ts

```
1  import { TestBed, async, inject } from '@angular/core/testing';
2
3  import { AuthGuard } from './auth.guard';
4
5  describe('AuthGuard', () => {
6    beforeEach(() => {
7      TestBed.configureTestingModule({
8        providers: [AuthGuard]
9      });
10   });
11
12   it('should ...', inject([AuthGuard], (guard: AuthGuard) => {
13     expect(guard).toBeTruthy();
14   }));
15 });
```

Arquivo 69: codigofonte/client/app/_guards/auth.guard.spec.ts

```
1  import { Injectable } from '@angular/core';
2  import { Router, CanActivate, ActivatedRouteSnapshot,
       RouterStateSnapshot } from '@angular/router';
3  import { EventService } from '../_services/event/event.service';
4
```

```
@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private router: Router, private eventService:
      EventService) { }

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot
      ) {
    if (localStorage.getItem('currentUser')) {
      this.eventService.setIsLogged(true);
      // logged in so return true
      return true;
    }
    // not logged in so redirect to login page with the return url
    this.router.navigate(['/login'], { queryParams: { returnUrl:
        state.url }});
    return false;
  }
}
```

Arquivo 70: codigofonte/client/app/_guards/auth.guard.ts

```
export class User {

  _id: string;
  username: string;
  password: string;
  email: string;

}
```

Arquivo 71: codigofonte/client/app/_models/user.ts

```
export class AuthenticationParam {
  authDatabase:string;
  userName:string;
  password:string;
  authMechanism:string;
  constructor(){
    this.authMechanism = "SCRAM-SHA-1"
  }
}
```

Arquivo 72: codigofonte/client/app/_params/authentication–param.ts

```
import { Headers, RequestOptions } from '@angular/http';
export class AuthorizationParam extends RequestOptions {
  constructor() {
    super();
    const currentUser = JSON.parse(localStorage.getItem('currentUser')
        );
        if (currentUser && currentUser.token) {
          this.headers = new Headers({ 'Authorization': `Bearer ${
              currentUser.token}` });
```

```
        }
  }
}
```

Arquivo 73: codigofonte/client/app/_params/authorization–param.ts

```
import { AuthenticationParam } from './authentication-param';
export class DatabaseParam {
  address:string;
  port:string;
  authentication:AuthenticationParam;
  userId:string;
  databaseName:string;
  collectionName:string;
  constructor() {
    this.authentication = new AuthenticationParam();
  }
}
```

Arquivo 74: codigofonte/client/app/_params/database–param.ts

```
export * from './authentication-param';
export * from './authorization-param';
export * from './database-param';
```

Arquivo 75: codigofonte/client/app/_params/params.ts

```
import { PrettyJsonPipe } from './pretty-json.pipe';

describe('PrettyJsonPipe', () => {
  it('create an instance', () => {
    const pipe = new PrettyJsonPipe();
    expect(pipe).toBeTruthy();
  });
});
```

Arquivo 76: codigofonte/client/app/_pipes/pretty–json.pipe.spec.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'prettyJson'
})
export class PrettyJsonPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    if(value){
      return JSON.stringify(value, null, 2)
      .replace(/ /g, ' ')
      .replace(/\n/g, '<br/>');
    }
  }

```

```
}
```

Arquivo 77: codigofonte/client/app/_pipes/pretty–json.pipe.ts

```
import { TestBed, inject } from '@angular/core/testing';

import { AlertService } from './alert.service';

describe('AlertService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [AlertService]
    });
  });

  it('should be created', inject([AlertService], (service:
      AlertService) => {
    expect(service).toBeTruthy();
  }));
});
```

Arquivo 78: codigofonte/client/app/_services/alert/alert.service.spec.ts

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { LoadingService } from '../loading/loading.service';
import { AuthorizationParam } from '../../_params/params';

@Injectable()
export class AlertService {

  private baseUrl: string = "/api/";
  private allAlertsUrl: string = this.baseUrl+"alerts";
  private allAlertsCountUrl: string = this.allAlertsUrl+"/count";

  constructor(private http: Http, private loadingService:
      LoadingService) { }

  listAlerts(){
    this.loadingService.show();
    return this.http.get(this.allAlertsUrl, this.jwt())
      .map((response: Response) => response.json())
      .finally(() => {this.loadingService.hide()});
  }

  countAlerts(){
    this.loadingService.show();
    return this.http.get(this.allAlertsCountUrl, this.jwt())
      .map((response: Response) => response.json())
      .finally(() => {this.loadingService.hide()});
  }

  deleteAlert(alertId){
```

```
30      this.loadingService.show();
31      return this.http.delete(`/api/alert/${alertId}`, this.jwt())
32        .map((response: Response) => "OK")
33        .finally(() => this.loadingService.hide());
34    }
35
36    private jwt() {
37      return new AuthorizationParam();
38    }
39
40  }
```

Arquivo 79: codigofonte/client/app/_services/alert/alert.service.ts

```
1  import { TestBed, inject } from '@angular/core/testing';
2
3  import { AuthenticationService } from './authentication.service';
4
5  describe('AuthenticationService', () => {
6    beforeEach(() => {
7      TestBed.configureTestingModule({
8        providers: [AuthenticationService]
9      });
10   });
11
12   it('should be created', inject([AuthenticationService], (service:
         AuthenticationService) => {
13     expect(service).toBeTruthy();
14   }));
15 });
```

Arquivo 80: codigofonte/client/app/_services/authentication/authentication.service.spec.ts

```
1  import { Injectable } from '@angular/core';
2  import { Http, Response } from '@angular/http';
3  import { Observable } from 'rxjs/Observable';
4  import 'rxjs/add/operator/map'
5  import { LoadingService } from '../loading/loading.service';
6
7  @Injectable()
8  export class AuthenticationService {
9
10   private baseUrl: string = "/api/";
11   private loginUrl: string = this.baseUrl+"login";
12
13   constructor(private http: Http, private loadingService:
         LoadingService) { }
14
15   login(email: string, password: string) {
16     this.loadingService.show();
17     return this.http.post(this.loginUrl, { 'email': email, 'password':
           password })
18       .map((response: Response) => {
```

```
19          // login successful if there's a jwt token in the response
20          const user = response.json();
21          if (user && user.token) {
22            // store user details and jwt token in local storage to keep
                  user logged in between page refreshes
23            localStorage.setItem('currentUser', JSON.stringify(user));
24          }
25        }).finally(() => this.loadingService.hide());
26    }
27
28    logout() {
29      // remove user from local storage to log user out
30      localStorage.removeItem('currentUser');
31    }
32
33  }
```

Arquivo 81: codigofonte/client/app/_services/authentication/authentication.service.ts

```
1  import { TestBed, inject } from '@angular/core/testing';
2
3  import { EventService } from './event.service';
4
5  describe('EventService', () => {
6    beforeEach(() => {
7      TestBed.configureTestingModule({
8        providers: [EventService]
9      });
10   });
11
12   it('should be created', inject([EventService], (service:
         EventService) => {
13     expect(service).toBeTruthy();
14   }));
15 });
```

Arquivo 82: codigofonte/client/app/_services/event/event.service.spec.ts

```
1  import { Injectable } from '@angular/core';
2  import { BehaviorSubject } from "rxjs/BehaviorSubject";
3  import { Observable } from "rxjs/Observable";
4
5  @Injectable()
6  export class EventService {
7
8    private isLogged: BehaviorSubject<boolean> = new BehaviorSubject<
         boolean>(null);
9      isLoggedEmitter: Observable<boolean> = this.isLogged.asObservable
           ();
10
11   constructor() { }
12
13   setIsLogged(isLogged: boolean) {
```

```
14          this.isLogged.next(isLogged);
15      }
16
17  }
```

Arquivo 83: codigofonte/client/app/_services/event/event.service.ts

```
1   import { TestBed, inject } from '@angular/core/testing';
2
3   import { FeedbackService } from './feedback.service';
4
5   describe('FeedbackService', () => {
6     beforeEach(() => {
7       TestBed.configureTestingModule({
8         providers: [FeedbackService]
9       });
10    });
11
12    it('should be created', inject([FeedbackService], (service:
          FeedbackService) => {
13      expect(service).toBeTruthy();
14    }));
15  });
```

Arquivo 84: codigofonte/client/app/_services/feedback/feedback.service.spec.ts

```
1   import { Injectable } from '@angular/core';
2   import { Router, NavigationStart } from '@angular/router';
3   import { Observable } from 'rxjs';
4   import { Subject } from 'rxjs/Subject';
5
6   @Injectable()
7   export class FeedbackService {
8
9       private subject: Subject<any> = new Subject<any>();
10    private keepAfterNavigationChange: boolean = false;
11
12    constructor(private router: Router) {
13      // clear alert message on route change
14          router.events.subscribe(event => {
15              if (event instanceof NavigationStart) {
16                  if (this.keepAfterNavigationChange) {
17                      // only keep for a single location change
18                      this.keepAfterNavigationChange = false;
19                  } else {
20                      // clear alert
21                      this.subject.next();
22                  }
23              }
24          });
25    }
26
27    success(message: string, keepAfterNavigationChange = false) {
```

```
28          this.keepAfterNavigationChange = keepAfterNavigationChange;
29          this.subject.next({ type: 'success', text: message });
30      }
31
32      error(message: string, keepAfterNavigationChange = false) {
33          this.keepAfterNavigationChange = keepAfterNavigationChange;
34          this.subject.next({ type: 'error', text: message });
35      }
36
37      getMessage(): Observable<any> {
38          return this.subject.asObservable();
39      }
40
41  }
```

Arquivo 85: codigofonte/client/app/_services/feedback/feedback.service.ts

```
1   import { TestBed, inject } from '@angular/core/testing';
2
3   import { JsonSchemaService } from './json-schema.service';
4
5   describe('JsonSchemaService', () => {
6     beforeEach(() => {
7       TestBed.configureTestingModule({
8         providers: [JsonSchemaService]
9       });
10    });
11
12    it('should be created', inject([JsonSchemaService], (service:
          JsonSchemaService) => {
13      expect(service).toBeTruthy();
14    }));
15  });
```

Arquivo 86: codigofonte/client/app/_services/json-schema/json-schema.service.spec.ts

```
1   import { Injectable } from '@angular/core';
2   import { Http, Headers, RequestOptions, Response } from '@angular/http
        ';
3   import { AuthorizationParam, DatabaseParam } from '../../_params/
        params';
4   import { LoadingService } from '../loading/loading.service';
5
6   @Injectable()
7   export class JsonSchemaService {
8
9     constructor(private http: Http, private loadingService:
          LoadingService) { }
10
11    discovery(databaseParam: DatabaseParam) {
12      this.loadingService.show();
13      return this.http.post('/api/batch/rawschema/steps/all',
            databaseParam, this.jwt())
```

```
14        .map((response: Response) => response.json());
15    }
16
17    listBatches(){
18      this.loadingService.show();
19      return this.http.get('/api/batches', this.jwt())
20        .map((response: Response) => response.json())
21        .finally(() => this.loadingService.hide());
22    }
23
24    deleteBatch(batchId){
25      this.loadingService.show();
26      return this.http.delete(`/api/batch/${batchId}`, this.jwt())
27        .map((response: Response) => "OK")
28        .finally(() => this.loadingService.hide());
29    }
30
31    getBatchById(batchId){
32      this.loadingService.show();
33      return this.http.get(`/api/batch/${batchId}`, this.jwt())
34        .map((response: Response) => response.json())
35        .finally(() => this.loadingService.hide());
36    }
37
38    getJsonSchemaByBatchId(batchId){
39      this.loadingService.show();
40      return this.http.get(`/api/batch/jsonschema/generate/${batchId}`,
          this.jwt())
41        .map((response: Response) => response.json())
42        .finally(() => this.loadingService.hide());
43    }
44
45    private jwt() {
46      return new AuthorizationParam();
47    }
48
49  }
```

Arquivo 87: codigofonte/client/app/_services/json–schema/json–schema.service.ts

```
1   import { TestBed, inject } from '@angular/core/testing';
2
3   import { LoadingService } from './loading.service';
4
5   describe('LoadingService', () => {
6     beforeEach(() => {
7       TestBed.configureTestingModule({
8         providers: [LoadingService]
9       });
10    });
11
12    it('should be created', inject([LoadingService], (service:
          LoadingService) => {
13      expect(service).toBeTruthy();
```

```
14    }));
15  });
```

Arquivo 88: codigofonte/client/app/_services/loading/loading.service.spec.ts

```
1   import { Injectable } from '@angular/core';
2   import { Observable, Subject } from 'rxjs/Rx';
3
4   export interface LoadingState {
5     show: boolean
6   }
7
8   @Injectable()
9   export class LoadingService {
10
11    private loadingSubject = new Subject();
12    loadingState = <Observable<LoadingState>>this.loadingSubject;
13
14    constructor() { }
15
16    show() {
17      this.loadingSubject.next(<LoadingState>{ show: true });
18    }
19
20    hide() {
21      this.loadingSubject.next(<LoadingState>{ show: false });
22    }
23
24  }
```

Arquivo 89: codigofonte/client/app/_services/loading/loading.service.ts

```
1   import { TestBed, inject } from '@angular/core/testing';
2
3   import { RegistrationService } from './registration.service';
4
5   describe('RegistrationService', () => {
6     beforeEach(() => {
7       TestBed.configureTestingModule({
8         providers: [RegistrationService]
9       });
10    });
11
12    it('should be created', inject([RegistrationService], (service:
          RegistrationService) => {
13      expect(service).toBeTruthy();
14    }));
15  });
```

Arquivo 90: codigofonte/client/app/_services/registration/registration.service.spec.ts

```
1   import { Injectable } from '@angular/core';
2   import { Http, Response } from '@angular/http';
```

```
3  import { User } from '../../_models/user';
4  import { LoadingService } from '../loading/loading.service';
5
6  @Injectable()
7  export class RegistrationService {
8
9      private baseUrl: string = "/api/";
10     private registerUrl: string = this.baseUrl+"register";
11
12   constructor(private http: Http, private loadingService:
         LoadingService) { }
13
14   register(user: User) {
15         this.loadingService.show();
16     return this.http.post(this.registerUrl, user)
17       .map((response: Response) => response.json())
18             .finally(() => this.loadingService.hide());
19   }
20 }
```

Arquivo 91: codigofonte/client/app/_services/registration/registration.service.ts

```
1  export * from './alert/alert.service';
2  export * from './authentication/authentication.service';
3  export * from './event/event.service';
4  export * from './feedback/feedback.service';
5  export * from './registration/registration.service';
6  export * from './json-schema/json-schema.service';
7  export * from './loading/loading.service';
8  export * from './user/user.service';
```

Arquivo 92: codigofonte/client/app/_services/services.ts

```
1  import { TestBed, inject } from '@angular/core/testing';
2
3  import { UserService } from './user.service';
4
5  describe('UserService', () => {
6    beforeEach(() => {
7      TestBed.configureTestingModule({
8        providers: [UserService]
9      });
10   });
11
12   it('should be created', inject([UserService], (service: UserService)
         => {
13     expect(service).toBeTruthy();
14   }));
15 });
```

Arquivo 93: codigofonte/client/app/_services/user/user.service.spec.ts

```
1  import { Injectable } from '@angular/core';
```

```
2  import { Http, Response } from '@angular/http';
3  import { User } from '../../_models/user';
4  import { LoadingService } from '../loading/loading.service';
5  import { AuthorizationParam } from '../../_params/params';
6
7
8  @Injectable()
9  export class UserService {
10
11     private baseUrl: string = "/api/user";
12
13   constructor(private http: Http, private loadingService:
       LoadingService) { }
14
15   get() {
16         this.loadingService.show();
17     return this.http.get(this.baseUrl, this.jwt())
18       .map((response: Response) => response.json())
19             .finally(() => this.loadingService.hide());
20   }
21
22   private jwt() {
23     return new AuthorizationParam();
24   }
25
26 }
```

Arquivo 94: codigofonte/client/app/_services/user/user.service.ts

```
1  import { NgModule }       from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3  import { AuthGuard }      from './_guards/auth.guard';
4  import {
5    AlertComponent,
6    DiscoveryComponent,
7    HomeComponent,
8    LoginComponent,
9    RegisterComponent,
10   UserComponent,
11   BatchComponent,
12   JsonSchemaComponent
13 } from './_components/components';
14
15 const routes: Routes = [
16   { path: '', component: HomeComponent, canActivate: [AuthGuard] },
17   { path: 'alert', component: AlertComponent, canActivate: [AuthGuard]
       },
18   { path: 'discovery', component: DiscoveryComponent, canActivate: [
       AuthGuard] },
19   { path: 'login', component: LoginComponent },
20     { path: 'register', component: RegisterComponent },
21   { path: 'user', component: UserComponent, canActivate: [AuthGuard]
       },
```

```
  { path: 'batch/:id', component: BatchComponent, canActivate: [
      AuthGuard] },
  { path: 'jsonschema/:id', component: JsonSchemaComponent,
      canActivate: [AuthGuard] },
    // otherwise redirect to home
    { path: '**', redirectTo: '' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Arquivo 95: codigofonte/client/app/app–routing.module.ts

Arquivo 96: codigofonte/client/app/app.component.css

```
<div>
    <app-navbar></app-navbar>
</div>
```

Arquivo 97: codigofonte/client/app/app.component.html

```
import { TestBed, async } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';

import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));

  it(`should have as title 'app'`, async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
  }));
```

```
29
30    it('should render title in a h1 tag', async(() => {
31      const fixture = TestBed.createComponent(AppComponent);
32      fixture.detectChanges();
33      const compiled = fixture.debugElement.nativeElement;
34      expect(compiled.querySelector('h1').textContent).toContain('
            Welcome to app!');
35    }));
36  });
```

Arquivo 98: codigofonte/client/app/app.component.spec.ts

```
1  import { Component } from '@angular/core';
2  import * as moment from 'moment';
3  import 'moment/min/locales';
4
5  @Component({
6    selector: 'app-root',
7    templateUrl: './app.component.html',
8    styleUrls: ['./app.component.css']
9  })
10 export class AppComponent {
11   constructor(){
12     moment.locale('pt-br');
13   }
14 }
```

Arquivo 99: codigofonte/client/app/app.component.ts

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { BrowserAnimationsModule } from '@angular/platform-browser/
       animations';
3  import { NgModule } from '@angular/core';
4  import { FormsModule, ReactiveFormsModule} from '@angular/forms';
5  import { HttpModule} from '@angular/http';
6  import { CdkTableModule} from '@angular/cdk/table';
7  import { MomentModule } from 'angular2-moment';
8  import {
9    MatAutocompleteModule,
10   MatButtonModule,
11   MatButtonToggleModule,
12   MatCardModule,
13   MatCheckboxModule,
14   MatChipsModule,
15   MatDatepickerModule,
16   MatDialogModule,
17   MatExpansionModule,
18   MatGridListModule,
19   MatIconModule,
20   MatInputModule,
21   MatListModule,
22   MatMenuModule,
23   MatNativeDateModule,
```

```
    MatPaginatorModule,
    MatProgressBarModule,
    MatProgressSpinnerModule,
    MatRadioModule,
    MatRippleModule,
    MatSelectModule,
    MatSidenavModule,
    MatSliderModule,
    MatSlideToggleModule,
    MatSnackBarModule,
    MatSortModule,
    MatTableModule,
    MatTabsModule,
    MatToolbarModule,
    MatTooltipModule,
    MatStepperModule,
    NoConflictStyleCompatibilityMode
} from '@angular/material';
import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { AuthGuard } from './_guards/auth.guard';
import {
    AlertService,
    AuthenticationService,
    EventService,
    FeedbackService,
    RegistrationService,
    JsonSchemaService,
    LoadingService,
    UserService
} from './_services/services';
import {
    FeedbackComponent,
    NavbarComponent,
    LoginComponent,
    RegisterComponent,
    HomeComponent,
    DiscoveryComponent,
    UserComponent,
    AlertComponent,
    SchemesComponent,
    BatchComponent,
    BatchDeleteModalComponent,
    JsonSchemaComponent,
    LoadingComponent,
    BatchElapsedTimeModalComponent
} from './_components/components';
import { PrettyJsonPipe } from './_pipes/pretty-json.pipe';

@NgModule({
    exports: [
        MatAutocompleteModule,
        MatButtonModule,
        MatButtonToggleModule,
```

```
78      MatCardModule,
79      MatCheckboxModule,
80      MatChipsModule,
81      MatStepperModule,
82      MatDatepickerModule,
83      MatDialogModule,
84      MatExpansionModule,
85      MatGridListModule,
86      MatIconModule,
87      MatInputModule,
88      MatListModule,
89      MatMenuModule,
90      MatNativeDateModule,
91      MatPaginatorModule,
92      MatProgressBarModule,
93      MatProgressSpinnerModule,
94      MatRadioModule,
95      MatRippleModule,
96      MatSelectModule,
97      MatSidenavModule,
98      MatSliderModule,
99      MatSlideToggleModule,
100     MatSnackBarModule,
101     MatSortModule,
102     MatTableModule,
103     MatTabsModule,
104     MatToolbarModule,
105     MatTooltipModule,
106     NoConflictStyleCompatibilityMode
107   ]
108 })
109 export class AppMaterialModule {}
110
111 @NgModule({
112   imports: [
113     BrowserModule,
114     BrowserAnimationsModule,
115     FormsModule,
116     HttpModule,
117     AppMaterialModule,
118     AppRoutingModule,
119     ReactiveFormsModule,
120     MomentModule
121   ],
122   declarations: [
123     AlertComponent,
124     DiscoveryComponent,
125     FeedbackComponent,
126     HomeComponent,
127     LoginComponent,
128     NavbarComponent,
129     RegisterComponent,
130     SchemesComponent,
131     UserComponent,
```

```
132       BatchComponent,
133       BatchDeleteModalComponent,
134       JsonSchemaComponent,
135       LoadingComponent,
136       BatchElapsedTimeModalComponent,
137       PrettyJsonPipe,
138       AppComponent
139     ],
140     providers: [
141       AuthGuard,
142       AlertService,
143       AuthenticationService,
144       RegistrationService,
145       EventService,
146       FeedbackService,
147       JsonSchemaService,
148       LoadingService,
149       UserService
150     ],
151     entryComponents: [
152       BatchDeleteModalComponent,
153       BatchElapsedTimeModalComponent
154     ],
155     bootstrap: [AppComponent]
156 })
157 export class AppModule { }
```

Arquivo 100: codigofonte/client/app/app.module.ts

```
1 export const environment = {
2   production: true
3 };
```

Arquivo 101: codigofonte/client/environments/environment.prod.ts

```
1 // The file contents for the current environment will overwrite these
      during build.
2 // The build system defaults to the dev environment which uses `
      environment.ts`, but if you do
3 // `ng build --env=prod` then `environment.prod.ts` will be used
      instead.
4 // The list of which env maps to which file can be found in `.angular-
      cli.json`.
5
6 export const environment = {
7   production: false
8 };
```

Arquivo 102: codigofonte/client/environments/environment.ts

```
1 <!doctype html>
2 <html lang="en">
3   <head>
```

```
    <meta charset="utf-8">
    <title>JSONSchemaDiscovery</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="./assets/favicon.ico" />
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

Arquivo 103: codigofonte/client/index.html

```
import './polyfills';
import 'hammerjs';
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Arquivo 104: codigofonte/client/main.ts

```
/**
 * This file includes polyfills needed by Angular and is loaded before the app.
 * You can add your own extra polyfills to this file.
 *
 * This file is divided into 2 sections:
 *   1. Browser polyfills. These are applied before loading ZoneJS and are sorted by browsers.
 *   2. Application imports. Files imported after ZoneJS that should be loaded before your main
 *      file.
 *
 * The current setup is for so-called "evergreen" browsers; the last versions of browsers that
 * automatically update themselves. This includes Safari >= 10, Chrome >= 55 (including Opera),
 * Edge >= 13 on the desktop, and iOS 10 and Chrome on mobile.
 *
 * Learn more in https://angular.io/docs/ts/latest/guide/browser-support.html
```

```
 */

/*
    ***************************************************************************
 * BROWSER POLYFILLS
 */

/** IE9, IE10 and IE11 requires all of the following polyfills. **/
// import 'core-js/es6/symbol';
// import 'core-js/es6/object';
// import 'core-js/es6/function';
// import 'core-js/es6/parse-int';
// import 'core-js/es6/parse-float';
// import 'core-js/es6/number';
// import 'core-js/es6/math';
// import 'core-js/es6/string';
// import 'core-js/es6/date';
// import 'core-js/es6/array';
// import 'core-js/es6/regexp';
// import 'core-js/es6/map';
// import 'core-js/es6/weak-map';
// import 'core-js/es6/set';

/** IE10 and IE11 requires the following for NgClass support on SVG
    elements */
// import 'classlist.js';  // Run `npm install --save classlist.js`.

/** Evergreen browsers require these. **/
import 'core-js/es6/reflect';
import 'core-js/es7/reflect';


/**
 * Required to support Web Animations `@angular/animation`.
 * Needed for: All but Chrome, Firefox and Opera. http://caniuse.com/#
     feat=web-animation
 **/
// import 'web-animations-js';  // Run `npm install --save web-
    animations-js`.



/*
    ***************************************************************************
 * Zone JS is required by Angular itself.
 */
import 'zone.js/dist/zone';  // Included with Angular CLI.


/*
    ***************************************************************************
```

```
 * APPLICATION IMPORTS
 */

/**
 * Date, currency, decimal and percent pipes.
 * Needed for: All but Chrome, Firefox, Edge, IE11 and Safari 10
 */
// import 'intl';  // Run `npm install --save intl`.
/**
 * Need to import at least one locale-data with intl.
 */
// import 'intl/locale-data/jsonp/en';
```

Arquivo 105: codigofonte/client/polyfills.ts

```
/* You can add global styles to this file, and also import other style
    files */
@import "~@angular/material/prebuilt-themes/indigo-pink.css";

html {
  height: 100%;
}

body {
  min-height: 100%;
}
```

Arquivo 106: codigofonte/client/styles.css

```
// This file is required by karma.conf.js and loads recursively all
    the .spec and framework files

import 'zone.js/dist/long-stack-trace-zone';
import 'zone.js/dist/proxy.js';
import 'zone.js/dist/sync-test';
import 'zone.js/dist/jasmine-patch';
import 'zone.js/dist/async-test';
import 'zone.js/dist/fake-async-test';
import { getTestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

// Unfortunately there's no typing for the `__karma__` variable. Just
    declare it as any.
declare const __karma__: any;
declare const require: any;

// Prevent Karma from running prematurely.
__karma__.loaded = function () {};

```

```
// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);
// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
// Finally, start Karma to run the tests.
__karma__.start();
```

Arquivo 107: codigofonte/client/test.ts

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app",
    "baseUrl": "./",
    "module": "es2015",
    "types": []
  },
  "exclude": [
    "test.ts",
    "**/*.spec.ts"
  ]
}
```

Arquivo 108: codigofonte/client/tsconfig.app.json

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/spec",
    "baseUrl": "./",
    "module": "commonjs",
    "target": "es5",
    "types": [
      "jasmine",
      "node"
    ]
  },
  "files": [
    "test.ts"
  ],
  "include": [
    "**/*.spec.ts",
    "**/*.d.ts"
  ]
}
```

Arquivo 109: codigofonte/client/tsconfig.spec.json

```
/* SystemJS module definition */
declare var module: NodeModule;
interface NodeModule {
  id: string;
}
```

Arquivo 110: codigofonte/client/typings.d.ts

```
import { TesteappPage } from './app.po';

describe('testeapp App', () => {
  let page: TesteappPage;

  beforeEach(() => {
    page = new TesteappPage();
  });

  it('should display welcome message', () => {
    page.navigateTo();
    expect(page.getParagraphText()).toEqual('Welcome to app!');
  });
});
```

Arquivo 111: codigofonte/e2e/app.e2e–spec.ts

```
import { browser, by, element } from 'protractor';

export class TesteappPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    return element(by.css('app-root h1')).getText();
  }
}
```

Arquivo 112: codigofonte/e2e/app.po.ts

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/e2e",
    "baseUrl": "./",
    "module": "commonjs",
    "target": "es5",
    "types": [
      "jasmine",
      "jasminewd2",
      "node"
    ]
  }
}
```

Arquivo 113: codigofonte/e2e/tsconfig.e2e.json

```
1 {"_id":{"$oid":"50319491fe4dce143835c552"},"binaryType":{"$binary":"
    dGVzdGFuZG8=","$type":"00"},"dateAsISO":{"$date":"2014-01-31
    T22:26:33.000Z"},"timestampType":{"$timestamp":{"t":1421006159,"i"
    :4}},"dbRefType":{"$ref":"some_collection","$id":"50319491
    fe4dce143835c552"},"undefinedType":{"$undefined":true},"longType"
    :{"$numberLong":"9223372036854775807"},"booleanType":false,"
    nullType":null,"regexType":{"$regex":"/^S/","$options":"m"},"
    stringType":"some_value","dateAsTime":{"$date":"2017-06-07
    T00:37:39.332Z"},"objectType":{"property":"some_value"},"
    arrayMultiValues":[48,"50",{"property":"teste"},[1,"some_value"],
    true,{"$binary":"dGVzdGFuZG8=","$type":"00"}],"arrayOfObject":[{"
    property":48},{"property":"some_value"}],"arrayOfString":["
    some_value","some_value"],"arrayOfNumber":[1,2],"arrayOfBoolean":[
    true,false],"arrayOfArray":[[1,2],["some_value","some_value"]],"
    arrayOfExtendedType":[{"$date":"2014-01-31T22:26:33.000Z"},{"
    $numberLong":"9223372036854775807"}],"javascriptType":{"$code":"
    var a = function(param){console.log(param);};"},"
    javascriptWithScopeType":{"$code":"int x = y","$scope":{"y":1}},"
    maxKeyType":{"$maxKey":1},"minKeyType":{"$minKey":1},"intType":1,"
    doubleType":12343.44,"intType2":1}
2 {"_id":{"$oid":"50319491fe4dce143835c553"},"binaryType":{"$binary":"
    dGVzdGFuZG8=","$type":"00"},"dateAsISO":{"$date":"2014-01-31
    T22:26:33.000Z"},"timestampType":{"$timestamp":{"t":1421006159,"i"
    :4}},"dbRefType":{"$ref":"some_collection","$id":"50319491
    fe4dce143835c552"},"undefinedType":{"$undefined":true},"longType"
    :{"$numberLong":"9223372036854775807"},"booleanType":false,"
    nullType":null,"regexType":{"$regex":"/^S/","$options":"m"},"
    stringType":"some_value","dateAsTime":{"$date":"2017-06-07
    T00:37:39.332Z"},"objectType":{"property":"some_value"},"
    arrayMultiValues":[48,"50",{"property":"teste"},[1,"some_value"],
    true],"arrayOfObject":[{"property":48}],"arrayOfString":["
    some_value","some_value"],"arrayOfNumber":[1,2],"arrayOfBoolean":[
    true,false],"arrayOfArray":[["some_value","some_value"]],"
    arrayOfExtendedType":[{"$date":"2014-01-31T22:26:33.000Z"},{"
    $numberLong":"9223372036854775807"}],"javascriptType":{"$code":"
    var a = function(param){console.log(param);};"},"
    javascriptWithScopeType":{"$code":"int x = y","$scope":{"y":1}},"
    maxKeyType":{"$maxKey":1},"minKeyType":{"$minKey":1},"intType":1,"
    doubleType":12343.44,"intType2":1}
3 {"_id":{"$oid":"50319491fe4dce143835c554"},"binaryType":{"$binary":"
    dGVzdGFuZG8=","$type":"00"},"dateAsISO":{"$date":"2014-01-31
    T22:26:33.000Z"},"timestampType":{"$timestamp":{"t":1421006159,"i"
    :4}},"dbRefType":{"$ref":"some_collection","$id":"50319491
    fe4dce143835c552"},"undefinedType":{"$undefined":true},"longType"
    :{"$numberLong":"9223372036854775807"},"booleanType":false,"
    nullType":null,"regexType":{"$regex":"/^S/","$options":"m"},"
    stringType":"some_value","dateAsTime":{"$date":"2017-06-07
    T00:37:39.332Z"},"maxKeyType":{"$maxKey":1},"minKeyType":{"$minKey
    ":1},"intType":1,"doubleType":12343.44,"intType2":1,"objectType":{
    "property":"some_value"},"arrayMultiValues":[48,"50",{"property":"
```

```
    teste"},[1,"some_value"],true],"arrayOfObject":[{"property":48}],"
    arrayOfString":["some_value","some_value"],"arrayOfNumber":[1,2],"
    arrayOfBoolean":[true,false],"arrayOfArray":[["some_value","
    some_value"]],"arrayOfExtendedType":[{"$date":"2014-01-31
    T22:26:33.000Z"},{"$numberLong":"9223372036854775807"}],"
    javascriptType":{"$code":"var a = function(param){console.log(
    param);};"},"javascriptWithScopeType":{"$code":"int x = y","$scope
    ":{"y":1}}}
{"_id":{"$oid":"50319491fe4dce143835c555"},"stringType":"some_value","
    dateAsTime":{"$date":"2017-06-07T00:37:39.332Z"},"maxKeyType":{"
    $maxKey":1},"minKeyType":{"$minKey":1},"intType":1,"doubleType"
    :12343.44,"intType2":1,"objectType":{"property":"some_value"},"
    arrayMultiValues":[48,"50",{"property":"teste"},[1,"some_value"],
    true],"arrayOfObject":[{"property":48}],"arrayOfString":["
    some_value","some_value"],"arrayOfNumber":[],"arrayOfBoolean":[
    true,false],"arrayOfArray":[["some_value","some_value"]],"
    arrayOfExtendedType":[{"$date":"2014-01-31T22:26:33.000Z"},{"
    $numberLong":"9223372036854775807"}],"javascriptType":{"$code":"
    var a = function(param){console.log(param);};"},"
    javascriptWithScopeType":{"$code":"int x = y","$scope":{"y":1}}}
{"_id":{"$oid":"50319491fe4dce143835c556"},"stringType":"other_value",
    "dateAsTime":{"$date":"2017-06-07T00:37:39.333Z"},"maxKeyType":{"
    $maxKey":1},"minKeyType":{"$minKey":1},"intType":1,"doubleType"
    :12.44,"intType2":2,"objectType":{"property":"other_value"},"
    arrayMultiValues":[48,"50",{"property":"teste"},[1,"some_value"],
    true],"arrayOfObject":[{"property":48}],"arrayOfString":["
    some_value","some_value"],"arrayOfNumber":[],"arrayOfBoolean":[
    true,false],"arrayOfArray":[["some_value","some_value"]],"
    arrayOfExtendedType":[{"$date":"2014-01-31T22:26:33.000Z"},{"
    $numberLong":"9223372036854775807"}],"javascriptType":{"$code":"
    var a = function(param){console.log(param);};"},"
    javascriptWithScopeType":{"$code":"int x = y","$scope":{"y":1}}}
```

Arquivo 114: codigofonte/examples/alltypes.json

```
{
  "_id": {
    "$oid": "50319491fe4dce143835c552"
  },
  "arrayMultiValues": [
    48,
    "50",
    {
      "property": "teste"
    },
    [1,2,4],
    true,
    {
     "$binary": "dGVzdGFuZG8=",
     "$type": "00"
    }
  ],
  "arrayOfObject": [
    {
```

```
      "property": 48
    },
    {
      "property": "teste"
    }
  ],
  "binaryType": {
    "$binary": "dGVzdGFuZG8=",
    "$type": "00"
  },
  "booleanType": false,
  "dateAsISO": {
    "$date": "2014-01-31T22:26:33.000Z"
  },
  "dateAsTime": {
    "$date": "2017-06-07T00:37:39.332Z"
  },
  "dbRefType": {
    "$id": "50319491fe4dce143835c552",
    "$ref": "users"
  },
  "doubleType": 12343.44,
  "intType": 1,
  "intType2": 1,
  "javascriptType": {
    "$code": "var a = function(param){console.log(param);};"
  },
  "javascriptWithScopeType": {
    "$code": "int x = y",
    "$scope": {
      "y": 1
    }
  },
  "longType": {
    "$numberLong": "9223372036854775807"
  },
  "maxKeyType": {
    "$maxKey": 1
  },
  "minKeyType": {
    "$minKey": 1
  },
  "nullType": null,
  "objectType": {
    "property": "value"
  },
  "regexType": {
    "$options": "m",
    "$regex": "/^S/"
  },
  "stringType": "Felipe de Souza da Costa",
  "timestampType": {
    "$timestamp": {
      "i": 4,
```

```
        "t": 1421006159
      }
    },
    "undefinedExample": {
      "$undefined": true
    }
}
```

Arquivo 115: codigofonte/examples/bsontypestest.json

```

{
    "_id": "ObjectID",
    "binaryType": "Binary",
    "dateAsISO": "Date",
    "timestampType": "Timestamp",
    "dbRefType": "DBRef",
    "undefinedType": "Undefined",
    "longType": "Long",
    "booleanType": "Boolean",
    "nullType": "Null",
    "regexType": "RegExp",
    "stringType": "String",
    "dateAsTime": "Date",
    "objectType": {
      "property": "String"
    },
    "arrayMultiValues": [
      "Number", "String",
      { "property": "String" },
      [ "Number", "String" ],
      "Boolean", "Binary"
    ],
    "arrayOfObject": [
      { "property": "Number" },
      { "property": "String" }
    ],
    "arrayOfString": [ "String" ],
    "arrayOfNumber": [ "Number" ],
    "arrayOfBoolean": [ "Boolean" ],
    "arrayOfArray": [ [ "Number" ],
      [ "String" ]
    ],
    "arrayOfExtendedType": [
      "Date",
      "Long"
    ],
    "javascriptType": "Code",
    "javascriptWithScopeType": "Code",
    "maxKeyType": "MaxKey",
    "minKeyType": "MinKey",
    "intType": "Number",
    "doubleType": "Number",
    "intType2": "Number"
```

```
}
```

Arquivo 116: codigofonte/examples/bsontypestest1rawschema.json

```

{
  "_id": "ObjectID",
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "dateAsTime": "Date",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number",
    "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean" ]
  ,
  "arrayOfObject": [
    { "property": "Number" }
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [ "Number" ],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number"
}
```

Arquivo 117: codigofonte/examples/bsontypestest2rawschema.json

```

{
  "_id": "ObjectID",
```

```
  "binaryType": "Binary",
  "dateAsISO": "Date",
  "timestampType": "Timestamp",
  "dbRefType": "DBRef",
  "undefinedType": "Undefined",
  "longType": "Long",
  "booleanType": "Boolean",
  "nullType": "Null",
  "regexType": "RegExp",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfObject": [
    { "property": "Number" }
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [ "Number" ],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [
    [ "String" ]
  ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ],
  "javascriptType": "Code",
  "javascriptWithScopeType": "Code"
}
```

Arquivo 118: codigofonte/examples/bsontypestest3rawschema.json

```

{
  "_id": "ObjectID",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
```

```
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ]
}
```

Arquivo 119: codigofonte/examples/bsontypestest4rawschema.json

```

{
  "_id": "ObjectID",
  "stringType": "String",
  "dateAsTime": "Date",
  "maxKeyType": "MaxKey",
  "minKeyType": "MinKey",
  "intType": "Number",
  "doubleType": "Number",
  "intType2": "Number",
  "objectType": {
    "property": "String"
  },
  "arrayMultiValues": [
    "Number", "String",
    { "property": "String" },
    [ "Number", "String" ],
    "Boolean"
  ],
  "arrayOfString": [ "String" ],
  "arrayOfNumber": [],
  "arrayOfBoolean": [ "Boolean" ],
  "arrayOfArray": [ [ "String" ] ],
  "arrayOfExtendedType": [
    "Date",
    "Long"
  ]
}
```

Arquivo 120: codigofonte/examples/bsontypestest5rawschema.json

```
 {
```

```
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "Binary": {
      "title": "Binary",
      "type": "object",
      "properties": {
        "$binary": {
          "type": "string"
        },
        "$type": {
          "type": "string"
        }
      },
      "required": [
        "$binary",
        "$type"
      ]
    },
    "Undefined": {
      "title": "Undefined",
      "type": "object",
      "properties": {
        "$undefined": {
          "type": "boolean"
        }
      },
      "required": [
        "$undefined"
      ]
    },
    "ObjectID": {
      "title": "ObjectID",
      "type": "object",
      "properties": {
        "$oid": {
          "type": "string"
        }
      },
      "required": [
        "$oid"
      ]
    },
    "Date": {
      "title": "Date",
      "type": "object",
      "properties": {
        "$date": {
          "type": "string"
        }
      },
      "required": [
        "$date"
      ]
    },
```

```
    "RegExp": {
      "title": "RegExp",
      "type": "object",
      "properties": {
        "$options": {
          "type": "string"
        },
        "$regex": {
          "$type": "string"
        }
      },
      "required": [
        "$options",
        "$regex"
      ]
    },
    "DBRef": {
      "title": "DBRef",
      "type": "object",
      "properties": {
        "$id": {
          "type": "string"
        },
        "$ref": {
          "type": "string"
        }
      },
      "required": [
        "$id",
        "$ref"
      ]
    },
    "Code": {
      "title": "Code",
      "type": "object",
      "properties": {
        "$code": {
          "type": "string"
        },
        "$scope": {
          "type": "object"
        }
      },
      "required": [
        "$code"
      ]
    },
    "Timestamp": {
      "title": "Timestamp",
      "type": "object",
      "properties": {
        "$timestamp": {
          "type": "object",
          "properties": {
```

```
            "i": {
              "type": "integer"
            },
            "t": {
              "type": "integer"
            }
          },
          "required": [
            "i",
            "t"
          ]
        }
      },
      "required": [
        "$timestamp"
      ]
    },
    "Long": {
      "title": "NumberLong",
      "type": "object",
      "properties": {
        "$numberLong": {
          "type": "string"
        }
      },
      "required": [
        "$numberLong"
      ]
    },
    "MinKey": {
      "title": "MinKey",
      "type": "object",
      "properties": {
        "$minKey": {
          "type": "integer"
        }
      },
      "required": [
        "$minKey"
      ]
    },
    "MaxKey": {
      "title": "MaxKey",
      "type": "object",
      "properties": {
        "$maxKey": {
          "type": "integer"
        }
      },
      "required": [
        "$maxKey"
      ]
    }
  },
```

```
  "properties": {
    "_id": {
      "$ref": "#/definitions/ObjectID"
    },
    "arrayMultiValues": {
      "name": "arrayMultiValues",
      "type": "array",
      "items": {
        "anyOf": [
          {
            "type": "number"
          },
          {
            "type": "string"
          },
          {
            "type": "object",
            "properties": {
              "property": {
                "name": "property",
                "type": "string"
              }
            },
            "additionalProperties": false,
            "required": [
              "property"
            ]
          },
          {
            "name": "arrayMultiValues",
            "type": "array",
            "items": {
              "anyOf": [
                {
                  "type": "number"
                },
                {
                  "type": "string"
                }
              ]
            },
            "minItems": 1,
            "additionalItems": true
          },
          {
            "type": "boolean"
          },
          {
            "$ref": "#/definitions/Binary"
          }
        ]
      },
      "minItems": 1,
      "additionalItems": true
```

```
    },
    "arrayOfArray": {
      "name": "arrayOfArray",
      "type": "array",
      "items": {
        "name": "arrayOfArray",
        "type": "array",
        "items": {
          "anyOf": [
            {
              "type": "string"
            },
            {
              "type": "number"
            }
          ]
        },
        "minItems": 1,
        "additionalItems": true
      },
      "minItems": 1,
      "additionalItems": true
    },
    "arrayOfBoolean": {
      "name": "arrayOfBoolean",
      "type": "array",
      "items": {
        "type": "boolean"
      },
      "minItems": 1,
      "additionalItems": true
    },
    "arrayOfExtendedType": {
      "name": "arrayOfExtendedType",
      "type": "array",
      "items": {
        "anyOf": [
          {
            "$ref": "#/definitions/Date"
          },
          {
            "$ref": "#/definitions/Long"
          }
        ]
      },
      "minItems": 1,
      "additionalItems": true
    },
    "arrayOfNumber": {
      "name": "arrayOfNumber",
      "type": "array",
      "items": {
        "type": "number"
      },
```

```
272         "minItems": 0,
273         "additionalItems": true
274       },
275       "arrayOfString": {
276         "name": "arrayOfString",
277         "type": "array",
278         "items": {
279           "type": "string"
280         },
281         "minItems": 1,
282         "additionalItems": true
283       },
284       "dateAsTime": {
285         "$ref": "#/definitions/Date"
286       },
287       "doubleType": {
288         "name": "doubleType",
289         "type": "number"
290       },
291       "intType": {
292         "name": "intType",
293         "type": "number"
294       },
295       "intType2": {
296         "name": "intType2",
297         "type": "number"
298       },
299       "maxKeyType": {
300         "$ref": "#/definitions/MaxKey"
301       },
302       "minKeyType": {
303         "$ref": "#/definitions/MinKey"
304       },
305       "objectType": {
306         "type": "object",
307         "properties": {
308           "property": {
309             "name": "property",
310             "type": "string"
311           }
312         },
313         "additionalProperties": false,
314         "required": [
315           "property"
316         ],
317         "name": "objectType"
318       },
319       "stringType": {
320         "name": "stringType",
321         "type": "string"
322       },
323       "arrayOfObject": {
324         "name": "arrayOfObject",
325         "type": "array",
```

```
      "items": {
        "type": "object",
        "properties": {
          "property": {
            "name": "property",
            "anyOf": [
              {
                "type": "number"
              },
              {
                "type": "string"
              }
            ]
          }
        },
        "additionalProperties": false,
        "required": [
          "property"
        ]
      },
      "minItems": 1,
      "additionalItems": true
    },
    "binaryType": {
      "$ref": "#/definitions/Binary"
    },
    "booleanType": {
      "name": "booleanType",
      "type": "boolean"
    },
    "dateAsISO": {
      "$ref": "#/definitions/Date"
    },
    "dbRefType": {
      "$ref": "#/definitions/DBRef"
    },
    "javascriptType": {
      "$ref": "#/definitions/Code"
    },
    "javascriptWithScopeType": {
      "$ref": "#/definitions/Code"
    },
    "longType": {
      "$ref": "#/definitions/Long"
    },
    "nullType": {
      "name": "nullType",
      "type": "null"
    },
    "regexType": {
      "$ref": "#/definitions/RegExp"
    },
    "timestampType": {
      "$ref": "#/definitions/Timestamp"
```

```
    },
    "undefinedType": {
      "$ref": "#/definitions/Undefined"
    }
  },
  "additionalProperties": false,
  "required": [
    "_id",
    "arrayMultiValues",
    "arrayOfArray",
    "arrayOfBoolean",
    "arrayOfExtendedType",
    "arrayOfNumber",
    "arrayOfString",
    "dateAsTime",
    "doubleType",
    "intType",
    "intType2",
    "maxKeyType",
    "minKeyType",
    "objectType",
    "stringType"
  ]
}
```

Arquivo 121: codigofonte/examples/bsontypestestjsonschema.json

```
{
  "fields": [
    {
      "name": "_id",
      "path": "_id",
      "count": 5,
      "types": [
        {
          "name": "ObjectID",
          "path": "_id",
          "count": 5
        }
      ]
    },
    {
      "name": "arrayMultiValues",
      "path": "arrayMultiValues",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayMultiValues",
          "count": 5,
          "items": [
            {
              "name": "Number",
              "path": "arrayMultiValues",
```

```
      "count": 5
    },
    {
      "name": "String",
      "path": "arrayMultiValues",
      "count": 5
    },
    {
      "name": "Object",
      "path": "arrayMultiValues",
      "count": 5,
      "fields": [
        {
          "name": "property",
          "path": "arrayMultiValues.property",
          "count": 5,
          "types": [
            {
              "name": "String",
              "path": "arrayMultiValues.property",
              "count": 5
            }
          ]
        }
      ]
    },
    {
      "name": "Array",
      "path": "arrayMultiValues",
      "count": 5,
      "items": [
        {
          "name": "Number",
          "path": "arrayMultiValues",
          "count": 5
        },
        {
          "name": "String",
          "path": "arrayMultiValues",
          "count": 5
        }
      ]
    },
    {
      "name": "Boolean",
      "path": "arrayMultiValues",
      "count": 5
    },
    {
      "name": "Binary",
      "path": "arrayMultiValues",
      "count": 1
    }
  ]
```

```
        }
      ]
    },
    {
      "name": "arrayOfArray",
      "path": "arrayOfArray",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfArray",
          "count": 5,
          "items": [
            {
              "name": "Array",
              "path": "arrayOfArray",
              "count": 5,
              "items": [
                {
                  "name": "String",
                  "path": "arrayOfArray",
                  "count": 5
                },
                {
                  "name": "Number",
                  "path": "arrayOfArray",
                  "count": 1
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "name": "arrayOfBoolean",
      "path": "arrayOfBoolean",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfBoolean",
          "count": 5,
          "items": [
            {
              "name": "Boolean",
              "path": "arrayOfBoolean",
              "count": 5
            }
          ]
        }
      ]
    },
    {
```

```
      "name": "arrayOfExtendedType",
      "path": "arrayOfExtendedType",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfExtendedType",
          "count": 5,
          "items": [
            {
              "name": "Date",
              "path": "arrayOfExtendedType",
              "count": 5
            },
            {
              "name": "Long",
              "path": "arrayOfExtendedType",
              "count": 5
            }
          ]
        }
      ]
    },
    {
      "name": "arrayOfNumber",
      "path": "arrayOfNumber",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfNumber",
          "count": 5,
          "items": [
            {
              "name": "Number",
              "path": "arrayOfNumber",
              "count": 3
            }
          ]
        }
      ]
    },
    {
      "name": "arrayOfString",
      "path": "arrayOfString",
      "count": 5,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfString",
          "count": 5,
          "items": [
            {
              "name": "String",
```

```
              "path": "arrayOfString",
              "count": 5
            }
          ]
        }
      ]
    },
    {
      "name": "dateAsTime",
      "path": "dateAsTime",
      "count": 5,
      "types": [
        {
          "name": "Date",
          "path": "dateAsTime",
          "count": 5
        }
      ]
    },
    {
      "name": "doubleType",
      "path": "doubleType",
      "count": 5,
      "types": [
        {
          "name": "Number",
          "path": "doubleType",
          "count": 5
        }
      ]
    },
    {
      "name": "intType",
      "path": "intType",
      "count": 5,
      "types": [
        {
          "name": "Number",
          "path": "intType",
          "count": 5
        }
      ]
    },
    {
      "name": "intType2",
      "path": "intType2",
      "count": 5,
      "types": [
        {
          "name": "Number",
          "path": "intType2",
          "count": 5
        }
      ]
```

```
    },
    {
      "name": "maxKeyType",
      "path": "maxKeyType",
      "count": 5,
      "types": [
        {
          "name": "MaxKey",
          "path": "maxKeyType",
          "count": 5
        }
      ]
    },
    {
      "name": "minKeyType",
      "path": "minKeyType",
      "count": 5,
      "types": [
        {
          "name": "MinKey",
          "path": "minKeyType",
          "count": 5
        }
      ]
    },
    {
      "name": "objectType",
      "path": "objectType",
      "count": 5,
      "types": [
        {
          "name": "Object",
          "path": "objectType",
          "count": 5,
          "fields": [
            {
              "name": "property",
              "path": "objectType.property",
              "count": 5,
              "types": [
                {
                  "name": "String",
                  "path": "objectType.property",
                  "count": 5
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "name": "stringType",
      "path": "stringType",
```

```
      "count": 5,
      "types": [
        {
          "name": "String",
          "path": "stringType",
          "count": 5
        }
      ]
    },
    {
      "name": "arrayOfObject",
      "path": "arrayOfObject",
      "count": 3,
      "types": [
        {
          "name": "Array",
          "path": "arrayOfObject",
          "count": 3,
          "items": [
            {
              "name": "Object",
              "path": "arrayOfObject",
              "count": 4,
              "fields": [
                {
                  "name": "property",
                  "path": "arrayOfObject.property",
                  "count": 4,
                  "types": [
                    {
                      "name": "Number",
                      "path": "arrayOfObject.property",
                      "count": 3
                    },
                    {
                      "name": "String",
                      "path": "arrayOfObject.property",
                      "count": 1
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "name": "binaryType",
      "path": "binaryType",
      "count": 3,
      "types": [
        {
          "name": "Binary",
```

```
            "path": "binaryType",
            "count": 3
          }
        ]
      },
      {
        "name": "booleanType",
        "path": "booleanType",
        "count": 3,
        "types": [
          {
            "name": "Boolean",
            "path": "booleanType",
            "count": 3
          }
        ]
      },
      {
        "name": "dateAsISO",
        "path": "dateAsISO",
        "count": 3,
        "types": [
          {
            "name": "Date",
            "path": "dateAsISO",
            "count": 3
          }
        ]
      },
      {
        "name": "dbRefType",
        "path": "dbRefType",
        "count": 3,
        "types": [
          {
            "name": "DBRef",
            "path": "dbRefType",
            "count": 3
          }
        ]
      },
      {
        "name": "javascriptType",
        "path": "javascriptType",
        "count": 3,
        "types": [
          {
            "name": "Code",
            "path": "javascriptType",
            "count": 3
          }
        ]
      },
      {
```

```
      "name": "javascriptWithScopeType",
      "path": "javascriptWithScopeType",
      "count": 3,
      "types": [
        {
          "name": "Code",
          "path": "javascriptWithScopeType",
          "count": 3
        }
      ]
    },
    {
      "name": "longType",
      "path": "longType",
      "count": 3,
      "types": [
        {
          "name": "Long",
          "path": "longType",
          "count": 3
        }
      ]
    },
    {
      "name": "nullType",
      "path": "nullType",
      "count": 3,
      "types": [
        {
          "name": "Null",
          "path": "nullType",
          "count": 3
        }
      ]
    },
    {
      "name": "regexType",
      "path": "regexType",
      "count": 3,
      "types": [
        {
          "name": "RegExp",
          "path": "regexType",
          "count": 3
        }
      ]
    },
    {
      "name": "timestampType",
      "path": "timestampType",
      "count": 3,
      "types": [
        {
          "name": "Timestamp",
```

```
            "path": "timestampType",
            "count": 3
          }
        ]
      },
      {
        "name": "undefinedType",
        "path": "undefinedType",
        "count": 3,
        "types": [
          {
            "name": "Undefined",
            "path": "undefinedType",
            "count": 3
          }
        ]
      }
    ],
    "count": 5
}
```

Arquivo 122: codigofonte/examples/bsontypestestrsus.json

```
import * as jwt from 'jsonwebtoken';
import UserController from '../controllers/user/user';
import RawSchemaController from '../controllers/rawSchema/rawSchema';
import RawSchemaBatchController from '../controllers/rawSchema/
    rawSchemaBatch';
import RawSchemaOrderedResultController from '../controllers/rawSchema
    /rawSchemaOrderedResult';
import RawSchemaUnionController from '../controllers/rawSchema/
    rawSchemaUnion';
import JsonSchemaExtractedController from '../controllers/jsonSchema/
    jsonSchemaExtracted';
import AlertController from '../controllers/alert/alert';

export default class ApiController {

  public login = (req, res) => {
      return new UserController().login(req.body.email,
          req.body.password).then((data) => {
        return this.success(res, data);
      }, (error) => {
        return this.error(res, error.message, error.code);
      });
    }

    public getUser = (req, res) => {
      const user = this.getUserByToken(req);
    if(user && user.user){
        return new UserController().getUser(user.user._id).then((data)
            => {
          return this.success(res, data);
        }, (error) => {
```

```
26            return this.error(res, error.message, error.code);
27          });
28      } else {
29        return this.error(res, "invalid token", 403);
30      }
31      }
32
33      public listBatchesByUserId = (req, res) => {
34        const user = this.getUserByToken(req);
35      if(user && user.user){
36        return new RawSchemaBatchController().listByUserId(user.user._id
              ).then((data) => {
37          return this.success(res, data);
38        }, (error) => {
39          return this.error(res, error, 500);
40        });
41      } else {
42        return this.error(res, "invalid token", 403);
43      }
44      }
45
46      public listAlertsByUserId = (req, res) => {
47        const user = this.getUserByToken(req);
48      if(user && user.user){
49        return new AlertController().listByUserId(user.user._id).then((
              data) => {
50          return this.success(res, data);
51        }, (error) => {
52          return this.error(res, error, 500);
53        });
54      } else {
55        return this.error(res, "invalid token", 403);
56      }
57      }
58
59      public countAlertsByUserId = (req, res) => {
60        const user = this.getUserByToken(req);
61      if(user && user.user){
62        return new AlertController().countByUserId(user.user._id).then((
              data) => {
63          return this.success(res, data);
64        }, (error) => {
65          return this.error(res, error, 500);
66        });
67      } else {
68        return this.error(res, "invalid token", 403);
69      }
70      }
71
72      public deleteAlert = (req, res) => {
73        const user = this.getUserByToken(req);
74      if(user && user.user){
75        return new AlertController().deleteAlert(req.params.id).then((
              data) => {
```

```
76          return this.success(res, data);
77        }, (error) => {
78          return this.error(res, error, 500);
79        });
80      } else {
81        return this.error(res, "invalid token", 403);
82      }
83      }
84
85    public allSteps = (req, res) => {
86      const user = this.getUserByToken(req);
87      if(user && user.user){
88        req.body.userId = user.user._id;
89        new RawSchemaBatchController().allSteps(req.body);
90          return this.success(res, "OK");
91      } else {
92        return this.error(res, "invalid token", 403);
93      }
94      }
95
96      public discovery = (req, res) => {
97      new RawSchemaBatchController().discovery(req.body);
98        return this.success(res, "OK");
99      }
100
101   public aggregate = (req, res) => {
102       return new RawSchemaBatchController().aggregate(req.body.batchId
              ).then((data) => {
103         return this.success(res, data);
104       }, (error) => {
105         return this.error(res, error.message, error.code);
106       });
107     }
108
109     public reduce = (req, res) => {
110       return new RawSchemaBatchController().mapReduce(req.body.batchId
              ).then((data) => {
111         return this.success(res, data);
112       }, (error) => {
113         return this.error(res, error.message, error.code);
114       });
115     }
116
117     public aggregateAndReduce = (req, res) => {
118       return new RawSchemaBatchController().aggregateAndReduce(
              req.body.batchId).then((data) => {
119         return this.success(res, data);
120       }, (error) => {
121         return this.error(res, error.message, error.code);
122       });
123     }
124
125     public union = (req, res) => {
126       return new RawSchemaOrderedResultController(req.body.batchId).
```

```
            union(req.body.batchId).then((data) => {
127           return this.success(res, data);
128         }, (error) => {
129         return this.error(res, error.message, error.code);
130         });
131     }
132
133     public generate = (req, res) => {
134         return new JsonSchemaExtractedController().generate(
                req.body.batchId).then((data) => {
135           return this.success(res, data);
136         }, (error) => {
137         return this.error(res, error.message, error.code);
138         });
139     }
140
141     public deleteBatch = (req, res) => {
142       return new RawSchemaBatchController().deleteBatch(req.params.id).
              then((data) => {
143           return this.success(res, data);
144         }, (error) => {
145         return this.error(res, error.message, error.code);
146         });
147     }
148
149     private error(res, err, code){
150       return res.status(code).json({ 'error': err });
151     }
152
153     private success(res, obj){
154       if(obj == null)
155         return res.sendStatus(200);
156       return res.status(200).json(obj);
157     }
158
159     private getUserByToken(req){
160       const authorization = req.headers.authorization;
161       if(!authorization)
162         return null;
163       const bearer = authorization.split("Bearer ");
164       if(bearer.length != 2)
165         return null;
166       const token = bearer[1];
167       try {
168         return jwt.verify(token, process.env.SECRET_TOKEN);
169       } catch(err) {
170         return null;
171       }
172     }
173 }
```

Arquivo 123: codigofonte/server/api/api.ts

```
1 import * as bodyParser from 'body-parser';
```

```
import * as dotenv from 'dotenv';
import * as express from 'express';
import * as morgan from 'morgan';
import * as mongoose from 'mongoose';
import * as path from 'path';
import * as http from 'http';
import * as fs from 'fs';
import * as rfs from 'rotating-file-stream';
import * as SegfaultHandler from 'segfault-handler';
import * as cluster from 'cluster';
import * as os from 'os';
import setRoutes from './routes';

const numCPUs = os.cpus().length;

if (cluster.isMaster) {
  console.log('Master process is running');
  // Fork workers
  for (let i = 0; i < numCPUs/2; i++) {
    cluster.fork();
  }
    // Listen for dying workers
  cluster.on('exit', function (worker) {
    // Replace the dead worker,
    // we're not sentimental
    console.log('Worker %d died :(', worker.id);
    cluster.fork();
  });
} else {
  const app = express();
  SegfaultHandler.registerHandler("crash.log");
  // Logging middleware
  // You can set morgan to log differently depending on your
      environment
  if (app.get('env') == 'production') {
    let morganLogDirectory = path.join(__dirname, '../../logs');
    // ensure log directory exists
    fs.existsSync(morganLogDirectory) || fs.mkdirSync(
        morganLogDirectory);
    // create a rotating write stream
    let morganLogStream = rfs('morgan.log', {
      interval: '1d', // rotate daily
      path: morganLogDirectory,
      size: '10M' // rotates the file when size exceeds 10 MegaBytes
    });
    // setup the logger
    app.use(morgan('combined', {stream: morganLogStream}))
  } else {
    // setup the logger
    app.use(morgan('dev'));
    // Load environment development variables
    dotenv.load({'path':'.env'});
  }

```

```
54    // Use body parser so we can get info from POST and/or URL
          parameters
55    app.use(bodyParser.json({limit: '50mb'}));
56    app.use(bodyParser.urlencoded({ extended: false }));
57
58    // Run the app by serving the static files in the dist directory
59    app.use(express.static(path.join(__dirname, '../public')));
60
61    // Set our api routes
62    setRoutes(app);
63
64    // For all GET requests, send back index.html so that
          PathLocationStrategy can be used
65    app.get('*', (req, res) => {
66      res.sendFile(path.join(__dirname, '../public/index.html'));
67    });
68
69    // Get port from environment and store in Express.
70    const port = Number(process.env.PORT || 3000);
71    app.set('port', port);
72
73    // Create HTTP server.
74    const server = http.createServer(app);
75
76    console.log(`Hello from Node.js ${cluster.isMaster ? 'master' : '
          child'} ${cluster.worker.id} process !\n`);
77
78    // Create a database variable outside of the database connection
          callback to reuse the connection pool in your app.
79    let db;
80
81    // Connect to the database before starting the application server.
82
83    mongoose.Promise = global.Promise;
84    mongoose.connect(process.env.MONGODB_URI, { useMongoClient: true },
          (err, database) => {
85      if (err) {
86        console.error.bind(console, 'connection error:')
87        process.exit(1);
88      }
89
90      // Save database object from the callback for reuse.
91      db = database;
92      console.log("Database connection ready");
93
94      mongoose.connection.db.admin().command({ setParameter: 1,
            failIndexKeyTooLong: false }).then((data) => {
95        console.log("setParameter done");
96      }).catch((error) => {
97        console.error("setParameter error",error);
98      });
99
100     // Listen on provided port, on all network interfaces.
101     try {
```

```
102       server.listen(port, () => console.log(`API running on
              localhost:${port}`));
103     } catch (err) {
104       console.log("GRAVE: ",err);
105     }
106
107   });
108
109 }
```

Arquivo 124: codigofonte/server/app.ts

```
1  import Alert from '../../models/alert/alert';
2  import BatchBaseController from '../batchBase';
3
4  export default class AlertController extends BatchBaseController {
5
6    model = Alert;
7
8    generate = (rawSchemaBatch): Promise<any> => {
9      return new Promise((resolv, reject) => {
10       this.model = new Alert({
11         "batchId":rawSchemaBatch._id,
12         "userId":rawSchemaBatch.userId,
13         "status":rawSchemaBatch.status,
14         "type":rawSchemaBatch.statusType,
15         "dbUri": rawSchemaBatch.dbUri,
16         "collectionName": rawSchemaBatch.collectionName,
17         "date": new Date()
18       });
19       this.model.save().then((data) => {
20         return resolv(data);
21       }).catch((error) => {
22         return reject({"type":"ALERT_GENERATE_ERROR", "message":
               error.message, "code":500});
23       });
24     });
25   }
26
27   listByUserId = (userId): Promise<any> => {
28     return new Promise((resolv, reject) => {
29       return this.model.find({"userId":userId}).sort("date").then((
             data) => {
30         return resolv(data);
31       }, (error) => {
32         return reject(error);
33       });
34     });
35   }
36
37   countByUserId = (userId): Promise<any> => {
38     return new Promise((resolv, reject) => {
39       return this.model.find({"userId":userId}).count().then((data) =>
             {
```

```
40          return resolv(data);
41        }, (error) => {
42          return reject(error);
43        });
44      });
45    }
46
47    deleteAlert = (alertId): Promise<any> => {
48      return new Promise((resolv, reject) => {
49        this.model.findOneAndRemove({ _id: alertId }).then((data) => {
50          return resolv(data);
51        }, (error) => {
52          return reject(error);
53        });
54      });
55    }
56
57
58  }
```

Arquivo 125: codigofonte/server/controllers/alert/alert.ts

```
1   abstract class BaseController {
2
3     abstract model: any;
4
5     // List all
6     public listAll = (req, res) => {
7       this.model.find({}, (err, docs) => {
8         if (err) { return this.error(res, err, 404); }
9         this.success(res, docs);
10      });
11    }
12
13    public listAllEntities = (): Promise<any> => {
14      return new Promise((resolv, reject) => {
15        this.model.find({}).then((data) => {
16          return resolv(data);
17        }).catch((error) => {
18          return reject({"type":"LIST_ALL_ERROR", "message":
                error.message, "code":404});
19        });
20      });
21    }
22
23    // Count all
24    public count = (req, res) => {
25      this.model.count((err, count) => {
26        if (err) { return this.error(res, err, 404); }
27        this.success(res, count);
28      });
29    }
30
31    public countAllEntities = (): Promise<any> => {
```

```
32      return new Promise((resolv, reject) => {
33        this.model.count().then((data) => {
34          return resolv(data);
35        }).catch((error) => {
36          return reject({"type":"COUNT_ALL_ERROR", "message":
                error.message, "code":404});
37        });
38      });
39    }
40
41    // Insert
42    public insert = (req, res) => {
43      const obj = new this.model(req.body);
44      obj.save((err, item) => {
45        // 11000 is the code for duplicate key error
46        if (err && err.code === 11000) { this.error(res, err, 400); }
47        if (err) { return this.error(res, err, 404); }
48        this.success(res, item);
49      });
50    }
51
52    public insertEntity = (entity): Promise<any> => {
53      return new Promise((resolv, reject) => {
54        const obj = new this.model(entity);
55        obj.save().then((data) => {
56          return resolv(data);
57        }).catch((error) => {
58          // 11000 is the code for duplicate key error
59          if (error.code === 11000) { return reject({"type":"
                DUPLICATE_KEY_ERROR", "message": error.message, "code"
                :400}); }
60          return reject({"type":"INSERT_ENTITY_ERROR", "message":
                error.message, "code":400});
61        });
62      });
63    }
64
65    public insertEntities = (entities): Promise<any> => {
66      return new Promise((resolv, reject) => {
67        this.model.insertMany(entities).then((data) => {
68          return resolv(data);
69        }).catch((error) => {
70          // 11000 is the code for duplicate key error
71          if (error.code === 11000) { return reject({"type":"
                DUPLICATE_KEY_ERROR", "message": error.message, "code"
                :400}); }
72          return reject({"type":"INSERT_ENTITY_ERROR", "message":
                error.message, "code":400});
73        });
74      });
75    }
76
77    // Get by id
78    public get = (req, res) => {
```

```
79      this.model.findOne({ '_id': req.params.id }, (err, obj) => {
80        if (err) { return this.error(res, err, 404); }
81        this.success(res, obj);
82      });
83    }
84
85    public getEntity = (id): Promise<any> => {
86      return new Promise((resolv, reject) => {
87        this.model.findOne({ '_id': id }).then((data) => {
88          return resolv(data);
89        }).catch((error) => {
90          return reject({"type":"GET_ENTITY_ERROR", "message":
                error.message, "code":404});
91        });
92      });
93    }
94
95    // Update by id
96    public update = (req, res) => {
97      this.model.findOneAndUpdate({ '_id': req.params.id }, {$inc: {
            __v: 1 } }, req.body, (err) => {
98        if (err) { return this.error(res, err, 404); }
99        this.success(res, null);
100     });
101   }
102
103   public updateEntity = (id, entity): Promise<any> => {
104     return new Promise((resolv, reject) => {
105       this.model.findOneAndUpdate({ '_id': id }, {$inc: { __v: 1 } },
              entity).then((data) => {
106         return resolv(data);
107       }).catch((error) => {
108         return reject({"type":"UPDATE_ENTITY_ERROR", "message":
                error.message, "code":404});
109       });
110     });
111   }
112
113   // Delete by id
114   public delete = (req, res) => {
115     this.model.findOneAndRemove({ _id: req.params.id }, (err) => {
116       if (err) { return this.error(res, err, 404); }
117       this.success(res, null);
118     });
119   }
120
121   public deleteEntity = (id): Promise<any> => {
122     return new Promise((resolv, reject) => {
123       this.model.findOneAndRemove({ _id: id }).then((data) => {
124         return resolv(data);
125       }).catch((error) => {
126         return reject({"type":"DELETE_ENTITY_ERROR", "message":
                error.message, "code":404});
127       });
```

```
128       });
129     }
130
131     public deleteAll = (): Promise<any> => {
132       return new Promise((resolv, reject) => {
133         this.model.remove({}).then((data) => {
134           return resolv(data);
135         }).catch((error) => {
136           return reject({"type":"REMOVE_ALL_ERROR", "message":
                  error.message, "code":404});
137         });
138       });
139     }
140
141     public error(res, err, code){
142       return res.status(code).json({ 'error': err });
143     }
144
145     public success(res, obj){
146       if(obj)
147         return res.status(200).json(obj);
148       return res.sendStatus(200);
149     }
150
151 }
152
153 export default BaseController;
```

Arquivo 126: codigofonte/server/controllers/base.ts

```
1  import BaseController from './base';
2
3  abstract class BatchBaseController extends BaseController {
4
5      listByBatchId = (req, res) => {
6        this.listEntitiesByBatchId(req.params.id).then((data) => {
7          return this.success(res, data);
8        }, (error) => {
9        return this.error(res, error, 404);
10       });
11     }
12
13     deleteByBatchId = (req, res) => {
14       this.deleteEntitiesByBatchId(req.params.id).then((data) => {
15         return this.success(res, "DELETED");
16       }, (error) => {
17         return this.error(res, error, 404);
18       });
19     }
20
21     countByBatchId = (req, res) => {
22       this.countEntitiesByBatchId(req.params.id).then((data) => {
23         return this.success(res, data);
24       }, (error) => {
```

```
25        return this.error(res, error, 404);
26      });
27    }
28
29    listEntitiesByBatchId = (batchId) => {
30      return this.model.find({'batchId':batchId});
31    }
32
33    deleteEntitiesByBatchId = (batchId) => {
34      return this.model.remove({'batchId': batchId});
35    }
36
37    countEntitiesByBatchId = (batchId) => {
38      return this.model.find({ 'batchId': batchId }).count();
39    }
40
41  }
42
43  export default BatchBaseController;
```

Arquivo 127: codigofonte/server/controllers/batchBase.ts

```
1   import JsonSchemaExtracted from '../../models/jsonSchema/
        jsonSchemaExtracted';
2   import BatchBaseController from '../batchBase';
3   import RawSchemaUnion from '../rawSchema/rawSchemaUnion';
4   import RawSchemaBatch from '../rawSchema/rawSchemaBatch';
5   import JsonSchemaBuilder from '../../helpers/jsonSchema/
        jsonSchemaBuilder';
6
7   export default class JsonSchemaExtractedController extends
        BatchBaseController {
8
9     model = JsonSchemaExtracted;
10
11    generate = (batchId): Promise<any> => {
12      return new Promise((resolv, reject) => {
13        let rawSchemaBatch;
14        new RawSchemaBatch().getById(batchId).then((data) => {
15          if(!data)
16            return reject({"message":`no results for batchId: ${batchId
                  }`,"code":404});
17          rawSchemaBatch = data;
18          return new RawSchemaUnion().listEntitiesByBatchId(batchId);
19        }).then((data) => {
20          return this.buildJsonSchema(data, batchId);
21        }).then((data) => {
22          rawSchemaBatch.endDate = new Date();
23          rawSchemaBatch.status = "DONE";
24          rawSchemaBatch.statusType = "DONE";
25          return rawSchemaBatch.save();
26        }).then((data) => {
27          return resolv(data);
28        }).catch((error) => {
```

```
29          return reject({"type":"MAPPER_JSONSCHEMA_ERROR", "message":
                error.message, "code":500});
30        });
31      });
32    }
33
34    private buildJsonSchema = (rawSchemaUnions, batchId): Promise<any>
          => {
35      return new Promise((resolv, reject) => {
36        const rawSchemaUnion = rawSchemaUnions.pop();
37        if (!rawSchemaUnion)
38          throw `no results for batchId: ${batchId}`;
39        const rawSchemaFinal = JSON.parse(rawSchemaUnion.rawSchemaFinal)
              ;
40        const jsonSchemaGenerated = new JsonSchemaBuilder().build(
              rawSchemaFinal.fields, rawSchemaFinal.count);
41        const jsonSchemaExtracted = new JsonSchemaExtracted({
42          batchId: batchId,
43          jsonSchema: JSON.stringify(jsonSchemaGenerated)
44        });
45        jsonSchemaExtracted.save().then((data) => {
46          return resolv(data);
47        }).catch((error) => {
48          return reject(error);
49        });
50      });
51    }
52
53  }
```

Arquivo 128: codigofonte/server/controllers/jsonSchema/jsonSchemaExtracted.ts

```
1   import * as mongoose from 'mongoose';
2   import {ObjectId}           from 'mongodb';
3   import rawSchemaSchema      from '../../models/rawSchema/rawSchema';
4   import options              from '../../params/mapReduceParam';
5   import BaseController  from '../base';
6
7   declare var emit;
8
9   export default class RawSchemaController extends BaseController {
10
11    model = null;
12
13    constructor(batchId: String) {
14      super();
15      const rawSchemaCollectionName = `rawSchema${batchId}`;
16      const RawSchema = mongoose.model(rawSchemaCollectionName,
            rawSchemaSchema);
17      this.model = RawSchema;
18    }
19
20    saveAll = (rawSchemas, batchId): Promise<any> => {
21      return new Promise((resolv, reject) => {
```

```
22        this.model.insertMany(rawSchemas, { ordered: true }).then((data)
              => {
23          return resolv(data);
24        }).catch((error) => {
25          return reject(error);
26        });
27      });
28    }
29
30    aggregate = (batchId): Promise<any> => {
31      return new Promise((resolv, reject) => {
32        this.model.aggregate([
33          { $project: { batchId: 1 , docRawSchema: 1, value:1 } },
34          { $group: { _id:"$docRawSchema", value:{$sum:1}, batchId: {
                $last: "$batchId" }, docRawSchema: { $last: "$docRawSchema
                " } } },
35          { $out: `rawschemaunordered${batchId}results` }
36        ]).allowDiskUse(true).exec().then((data) => {
37          return resolv(data);
38        }).catch((error) => {
39          console.error("error",error);
40          return reject(error);
41        });
42      });
43    }
44
45    mapReduce = (batchId): Promise<any> => {
46      return new Promise((resolv, reject) => {
47        options.out = { 'replace':`rawschemaunordered${batchId}results`
              };
48        options.map = function() {
49          emit(this.docRawSchema, 1);
50        };
51        options.reduce = function(key, values) {
52          let count = 0;
53          values.forEach((value) => {
54            count += value;
55          });
56          return count;
57        };
58        this.model.mapReduce(options).then((data) => {
59          return resolv(data);
60        }).catch((error) => {
61           console.error("error",error);
62           return reject(error);
63        });
64      });
65    }
66
67  }
```

Arquivo 129: codigofonte/server/controllers/rawSchema/rawSchema.ts

```
1  import * as mongodb from 'mongodb';
```

```
2  import {MongoClient} from 'mongodb'
3  import RawSchemaBatch from '../../models/rawSchema/rawSchemaBatch';
4  import Executor from '../../helpers/rawSchema/executor';
5  import DatabaseParam from '../../params/databaseParam';
6  import BaseController from '../base';
7  import RawSchemaController from './rawSchema';
8  import RawSchemaOrderedResultController from './rawSchemaOrderedResult
       ';
9  import RawSchemaUnorderedResultController from './
       rawSchemaUnorderedResult';
10 import RawSchemaUnionController from './rawSchemaUnion';
11 import JsonSchemaExtractedController from '../jsonSchema/
       jsonSchemaExtracted';
12 import AlertController from '../alert/alert'
13
14 export default class RawSchemaBatchController extends BaseController {
15
16   model = RawSchemaBatch;
17
18   allSteps = (params): Promise<any> => {
19     return new Promise((resolv, reject) => {
20
21       const databaseParam = new DatabaseParam(JSON.parse(
             JSON.stringify(params)));
22       const rawSchemaBatch = new this.model();
23       rawSchemaBatch.collectionName = databaseParam.collectionName;
24       rawSchemaBatch.dbUri = databaseParam.getURIWithoutAuthentication
             ();
25       rawSchemaBatch.userId = databaseParam.userId;
26       rawSchemaBatch.startDate = new Date();
27       rawSchemaBatch.status = "CONNECT_DATABASE";
28       rawSchemaBatch.save().then((data) => {
29         return this.connect(databaseParam);
30       }).then((data) => {
31         console.log("CONNECT: DONE");
32         return this.getCollection(data, rawSchemaBatch);
33       }).then((data) => {
34         console.log("GET COLLECTION: DONE");
35         return this.executeStepOne(data, rawSchemaBatch);
36       }).then((data) => {
37         console.log("STEP1: DONE");
38         return this.executeStepTwo(rawSchemaBatch);
39       }).then((data) => {
40         console.log("STEP2: DONE");
41         return this.executeStepThree(rawSchemaBatch);
42       }).then((data) => {
43         console.log("STEP3: DONE");
44         return this.executeStepFour(rawSchemaBatch);
45       }).then((data) => {
46         console.log("STEP4: DONE");
47         return this.generateAlert(data);
48       }).then((data) => {
49         console.log("ALERT: DONE");
50         return resolv(data);
```

```
51        }).catch((error) => {
52          console.error("STEPS error",error)
53          if(rawSchemaBatch){
54            rawSchemaBatch.status = "ERROR";
55            rawSchemaBatch.statusType = error.type;
56            rawSchemaBatch.statusMessage = error.message;
57            rawSchemaBatch.save().then((data) => {
58              return this.generateAlert(rawSchemaBatch);
59            }).then((data) => {
60              return resolv(data);
61            }).catch((error) => {
62              return reject(error);
63            });
64          } else {
65            return reject(error);
66          }
67        });
68      });
69    }
70
71    deleteBatch = (batchId): Promise<any> => {
72      return new Promise((resolv, reject) => {
73        this.model.findOneAndRemove({ _id: batchId }).then((data) =>{
74          return new RawSchemaController(batchId).deleteAll();
75        }).then((data) => {
76          return new RawSchemaOrderedResultController(batchId).deleteAll
                ();
77        }).then((data) => {
78          return new RawSchemaUnorderedResultController(batchId).
                deleteAll();
79        }).then((data) => {
80          return new RawSchemaUnionController().deleteEntitiesByBatchId(
                batchId);
81        }).then((data) => {
82          return new JsonSchemaExtractedController().
                deleteEntitiesByBatchId(batchId);
83        }).then((data) => {
84          return new AlertController().deleteEntitiesByBatchId(batchId);
85        }).then((data) => {
86          return resolv(data);
87        }).catch((error) => {
88          return reject(error);
89        });
90      });
91    }
92
93    discovery = (params): Promise<any> => {
94      return new Promise((resolv, reject) => {
95
96        const databaseParam = new DatabaseParam(JSON.parse(
              JSON.stringify(params)));
97        const rawSchemaBatch = new this.model();
98        rawSchemaBatch.collectionName = databaseParam.collectionName;
99        rawSchemaBatch.dbUri = databaseParam.getURIWithoutAuthentication
```

```
        ();
100       rawSchemaBatch.userId = databaseParam.userId;
101       rawSchemaBatch.startDate = new Date();
102
103       this.connect(databaseParam).then((data) => {
104         return this.getCollection(data, rawSchemaBatch);
105       }).then((data) => {
106         return this.executeStepOne(data, rawSchemaBatch);
107       }).then((data) => {
108         return resolv(data);
109       }).catch((error) => {
110         if(rawSchemaBatch){
111           rawSchemaBatch.status = "ERROR";
112           rawSchemaBatch.statusMessage = error;
113           rawSchemaBatch.save();
114         } else {
115           return reject(error);
116         }
117       });
118     });
119   }
120
121   private connect = (databaseParam): Promise<any> => {
122     return new Promise((resolv, reject) => {
123       MongoClient.connect(databaseParam.getURI(), { connectTimeoutMS:
              5000 }).then((database) => {
124         return resolv(database);
125       }).catch((error) => {
126         return reject({"type":"DATABASE_CONNECTION_ERROR", "message":
                error.message, "code":400});
127       });
128     });
129   }
130
131   private getCollection = (database, rawSchemaBatch): Promise<any> =>
        {
132     return new Promise((resolv, reject) => {
133       const collection = database.collection(
            rawSchemaBatch.collectionName);
134       collection.count().then((count) => {
135         if(count === 0)
136           return reject({"type":"EMPTY_COLLECTION_ERROR", "message": "
                coleção não encontrada", "code":400});
137         rawSchemaBatch.collectionCount = count;
138         rawSchemaBatch.status = "LOADING_DOCUMENTS";
139         return rawSchemaBatch.save();
140       }).then((data) => {
141         return resolv(collection);
142       }).catch((error) => {
143         return reject({"type":"EMPTY_COLLECTION_ERROR", "message": "
              coleção não encontrada", "code":400});
144       });
145     });
146   }
```

```
147
148     private executeStepOne = (collection, rawSchemaBatch): Promise<any>
            => {
149       return new Executor().execute(collection, rawSchemaBatch);
150     }
151
152     private executeStepTwo = (rawSchemaBatch) => {
153       // return this.mapReduce(rawSchemaBatch._id);
154       return this.aggregate(rawSchemaBatch._id);
155       // return this.aggregateAndReduce(rawSchemaBatch._id);
156     }
157
158     private executeStepThree = (rawSchemaBatch) => {
159       return new RawSchemaOrderedResultController(rawSchemaBatch._id).
              union(rawSchemaBatch._id);
160     }
161
162     private executeStepFour = (rawSchemaBatch) => {
163       return new JsonSchemaExtractedController().generate(
              rawSchemaBatch._id);
164     }
165
166     private generateAlert = (rawSchemaBatch) => {
167       return new AlertController().generate(rawSchemaBatch);
168     }
169
170     mapReduce = (batchId): Promise<any> => {
171       return new Promise((resolv, reject) => {
172         let rawSchemaBatch;
173         this.getById(batchId).then((rawSchemaBatchResult) => {
174           if (!rawSchemaBatchResult)
175             return reject({"message":`rawSchemaBatch with id: ${batchId}
                    not found`, "code":404});
176           return rawSchemaBatchResult;
177         }).then((data) => {
178           rawSchemaBatch = data;
179           rawSchemaBatch.reduceType = "MAP_REDUCE";
180           rawSchemaBatch.unorderedMapReduceDate = null;
181           rawSchemaBatch.orderedMapReduceDate = null;
182           rawSchemaBatch.unorderedAggregationDate = null;
183           rawSchemaBatch.orderedAggregationDate = null;
184           return new RawSchemaController(batchId).mapReduce(batchId);
185         }).then((data) => {
186           console.log("STEP2.1: DONE");
187           new RawSchemaUnorderedResultController(batchId).
                countAllEntities().then((data) => {
188             rawSchemaBatch.uniqueUnorderedCount = data;
189             rawSchemaBatch.unorderedMapReduceDate = new Date();
190             return rawSchemaBatch.save();
191           }).catch((error) => {
192             console.log("error",error);
193           })
194           return new RawSchemaUnorderedResultController(batchId).
                mapReduce(batchId);
```

```
195         }).then((data) => {
196           console.log("STEP2.2: DONE");
197           new RawSchemaOrderedResultController(batchId).countAllEntities
                ().then((data) => {
198             rawSchemaBatch.uniqueOrderedCount = data;
199             rawSchemaBatch.status = "UNION_DOCUMENTS";
200             rawSchemaBatch.orderedMapReduceDate = new Date();
201             return rawSchemaBatch.save();
202           }).catch((error) => {
203             console.log("error",error);
204           })
205           return resolv(data);
206         }).catch((error) => {
207           return reject({"type":"REDUCE_DOCUMENTS_ERROR", "message":
                error.message, "code":400});
208         });
209       });
210     }
211
212     aggregate = (batchId): Promise<any> => {
213       return new Promise((resolv, reject) => {
214         let rawSchemaBatch;
215         this.getById(batchId).then((rawSchemaBatchResult) => {
216           if (!rawSchemaBatchResult)
217             return reject({"message":`rawSchemaBatch with id: ${batchId}
                    not found`, "code":404});
218           return rawSchemaBatchResult;
219         }).then((data) => {
220           rawSchemaBatch = data;
221           rawSchemaBatch.reduceType = "AGGREGATE";
222           rawSchemaBatch.unorderedMapReduceDate = null;
223           rawSchemaBatch.orderedMapReduceDate = null;
224           rawSchemaBatch.unorderedAggregationDate = null;
225           rawSchemaBatch.orderedAggregationDate = null;
226           return new RawSchemaController(batchId).aggregate(batchId);
227         }).then((data) => {
228           console.log("STEP2.1: DONE");
229           new RawSchemaUnorderedResultController(batchId).
                countAllEntities().then((data) => {
230             rawSchemaBatch.uniqueUnorderedCount = data;
231             rawSchemaBatch.unorderedAggregationDate = new Date();
232             return rawSchemaBatch.save()
233           }).catch((error) => {
234             console.log("error",error);
235           });
236           return new RawSchemaUnorderedResultController(batchId).
                aggregate(batchId);
237         }).then((data) => {
238           console.log("STEP2.2: DONE");
239           new RawSchemaOrderedResultController(batchId).countAllEntities
                ().then((data) => {
240             rawSchemaBatch.uniqueOrderedCount = data;
241             rawSchemaBatch.status = "UNION_DOCUMENTS";
242             rawSchemaBatch.orderedAggregationDate = new Date();
```

```
243           return rawSchemaBatch.save();
244         }).catch((error) => {
245           console.log("error",error);
246         });
247         return resolv(data);
248       }).catch((error) => {
249         return reject({"type":"REDUCE_DOCUMENTS_ERROR", "message":
                error.message, "code":400});
250       });
251     });
252   }
253
254   aggregateAndReduce = (batchId): Promise<any> => {
255     return new Promise((resolv, reject) => {
256       let rawSchemaBatch;
257       this.getById(batchId).then((rawSchemaBatchResult) => {
258         if (!rawSchemaBatchResult)
259           return reject({"message":`rawSchemaBatch with id: ${batchId}
                not found`, "code":404});
260         return rawSchemaBatchResult;
261       }).then((data) => {
262         rawSchemaBatch = data;
263         rawSchemaBatch.reduceType = "AGGREGATE_AND_MAP_REDUCE";
264         rawSchemaBatch.unorderedMapReduceDate = null;
265         rawSchemaBatch.orderedMapReduceDate = null;
266         rawSchemaBatch.unorderedAggregationDate = null;
267         rawSchemaBatch.orderedAggregationDate = null;
268         return new RawSchemaController(batchId).aggregate(batchId);
269       }).then((data) => {
270         console.log("STEP2.1: DONE");
271         new RawSchemaUnorderedResultController(batchId).
              countAllEntities().then((data) => {
272           rawSchemaBatch.uniqueUnorderedCount = data;
273           rawSchemaBatch.unorderedAggregationDate = new Date();
274           return rawSchemaBatch.save();
275         }).catch((error) => {
276           console.log("error",error);
277         });
278           return new RawSchemaUnorderedResultController(batchId).
                mapReduce(batchId);
279         }).then((data) => {
280           console.log("STEP2.2: DONE");
281           new RawSchemaOrderedResultController(batchId).
                countAllEntities().then((data) => {
282             rawSchemaBatch.uniqueOrderedCount = data;
283             rawSchemaBatch.status = "UNION_DOCUMENTS";
284             rawSchemaBatch.orderedMapReduceDate = new Date();
285             return rawSchemaBatch.save();
286           }).catch((error) => {
287             console.log("error",error);
288           });
289           return resolv(rawSchemaBatch);
290       }).catch((error) => {
291         return reject({"type":"REDUCE_DOCUMENTS_ERROR", "message":
```

```
            error.message, "code":400});
292       });
293     });
294   }
295
296   getById = (id) => {
297     return this.model.findById(id);
298   }
299
300   listByUserId = (userId) => {
301     return this.model.find({"userId":userId});
302   }
303
304 }
```

Arquivo 130: codigofonte/server/controllers/rawSchema/rawSchemaBatch.ts

```
1  import * as mongoose from 'mongoose';
2  import rawSchemaOrderedResultSchema from '../../models/rawSchema/
       rawSchemaOrderedResult';
3  import BaseController from '../base';
4  import RawSchemaUnionController from './rawSchemaUnion';
5  import RawSchemaBatch from './rawSchemaBatch';
6
7  export default class RawSchemaOrderedResultController extends
       BaseController {
8
9    model = null;
10
11   constructor(batchId: String) {
12     super();
13     const rawSchemaOrderedResultCollectionName = `rawSchemaOrdered${
           batchId}Result`;
14     const RawSchemaOrderedResult = mongoose.model(
           rawSchemaOrderedResultCollectionName,
           rawSchemaOrderedResultSchema);
15     this.model = RawSchemaOrderedResult;
16   }
17
18   union = (batchId): Promise<any> => {
19     return new Promise((resolv, reject) => {
20       let rawSchemaBatch;
21       new RawSchemaBatch().getById(batchId).then((data) => {
22         if(!data)
23           return reject({"message":`no results for batchId: ${batchId
               }`,"code":404});
24         rawSchemaBatch = data;
25         return this.listAllEntities();
26       }).then((data) => {
27         return new RawSchemaUnionController().union(data, batchId);
28       }).then((data) => {
29         rawSchemaBatch.unionDate = new Date();
30         rawSchemaBatch.status = "MAPPER_JSONSCHEMA";
31         rawSchemaBatch.save().then((data) => {
```

```
32
33          }).catch((error => {
34
35          }));
36          return resolv(data);
37        }).catch((error) => {
38          console.error("union error",error);
39          return reject({"type":"UNION_DOCUMENTS_ERROR", "message":
                error.message, "code":500});
40        });
41      });
42    }
43
44  }
```

Arquivo 131: codigofonte/server/controllers/rawSchema/rawSchemaOrderedResult.ts

```
1  import RawSchemaUnion from '../../models/rawSchema/rawSchemaUnion';
2  import RawSchemaUnifier from '../../helpers/rawSchema/rawSchemaUnifier
       ';
3  import BatchBaseController from '../batchBase';
4
5  export default class RawSchemaUnionController extends
       BatchBaseController {
6
7    model = RawSchemaUnion;
8
9    union = (rawSchemaResults, batchId): Promise<any> => {
10     return new Promise((resolv, reject) => {
11       if(!rawSchemaResults || rawSchemaResults.length == 0)
12         throw `no results for batchId: ${batchId}`;
13       const rawSchemaFinal = new RawSchemaUnifier().union(
             rawSchemaResults);
14       const rawSchemaUnion = new RawSchemaUnion({
15         "batchId": batchId,
16         "rawSchemaFinal": JSON.stringify(rawSchemaFinal)
17       });
18       rawSchemaUnion.save().then((data) => {
19         resolv(data);
20       }, (error) => {
21         reject(error);
22       });
23     });
24   }
25
26 }
```

Arquivo 132: codigofonte/server/controllers/rawSchema/rawSchemaUnion.ts

```
1  import * as mongoose           from 'mongoose';
2  import {ObjectId}                  from 'mongodb';
3  import rawSchemaUnorderedResultSchema   from '../../models/rawSchema/
       rawSchemaUnorderedResult';
```

```
4  import RawSchemaOrderedResult   from '../../controllers/rawSchema/
       rawSchemaOrderedResult';
5  import BatchBaseController        from '../batchBase';
6  import ObjectKeysSorter         from '../../helpers/objectKeysSorter';
7  import rawSchemaOrder         from '../../helpers/rawSchema/
       rawSchemaOrder';
8  import options              from '../../params/mapReduceParam';
9
10 declare var emit;
11 declare var sort;
12 declare var docRawSchema;
13
14 export default class RawSchemaUnorderedResultController extends
       BatchBaseController {
15
16     model = null;
17
18   constructor(batchId: String) {
19     super();
20     const rawSchemaUnorderedResultCollectionName = `
           rawSchemaUnordered${batchId}Result`;
21     const RawSchemaUnorderedResult = mongoose.model(
           rawSchemaUnorderedResultCollectionName,
           rawSchemaUnorderedResultSchema);
22     this.model = RawSchemaUnorderedResult;
23   }
24
25   mapReduce = (batchId): Promise<any> => {
26     return new Promise((resolv, reject) => {
27       const sort = new ObjectKeysSorter().sort;
28         const sortObject = new ObjectKeysSorter().sortObject;
29         options.out = { 'replace':`rawschemaordered${batchId}results`
               };
30         options.map = function() {
31           const unorderedObject = this.docRawSchema ? JSON.parse(
                 this.docRawSchema) : JSON.parse(this._id);
32           const orderedObject = sort(unorderedObject);
33           const orderedObjectJson = JSON.stringify(orderedObject);
34             emit(orderedObjectJson, this.value);
35         };
36         options.reduce = function(key, values) {
37           let count = 0;
38           values.forEach((value) => {
39             count += value;
40           });
41           return count;
42         };
43         options.scope = { 'sort': sort, 'sortObject': sortObject, '
               batchId':batchId }
44       this.model.mapReduce(options).then((data) => {
45         return resolv(data);
46       }).catch((error) => {
47         console.error("error",error);
48         return reject(error);
```

```
49          });
50      });
51      }
52
53    aggregate = (batchId): Promise<any> => {
54      return new Promise((resolv, reject) => {
55        const collectionToWork = this.model.find({});
56        let quantity = -1;
57        let updateQuantity = 0;
58        collectionToWork.count().then((data) => {
59          quantity = data;
60        })
61        const order = collectionToWork.cursor().pipe(rawSchemaOrder());
62        order.on('progress', (data) => {
63          this.model.update(
64            { _id: data._id },
65            { "$set":{'docRawSchema': data.docRawSchema} }
66          ).then((data) => {
67            updateQuantity++;
68            if(quantity >= 0 && updateQuantity >= quantity){
69              this.executeAggregation(batchId).then((data) => {
70                return resolv(data);
71              }).catch((error) => {
72                console.error("error",error);
73                return reject(error);
74              });
75            }
76          }).catch((error) => {
77            console.error("error",error);
78            return reject(error);
79          });
80        });
81        order.on('ignore', () => {
82          console.log("ignore");
83          updateQuantity++;
84          if(quantity >= 0 && updateQuantity >= quantity){
85            this.executeAggregation(batchId).then((data) => {
86              return resolv(data);
87            }).catch((error) => {
88              console.error("error",error);
89              return reject(error);
90            });
91          }
92        });
93        order.on('error',(error) => {
94          console.error("error",error);
95          reject(error);
96        });
97      });
98    }
99
100   executeAggregation = (batchId): Promise<any> => {
101     return new Promise((resolv, reject) => {
102       this.model.aggregate([
```

```
103         { $project: { docRawSchema: 1, value:1 } },
104         { $group: { _id:"$docRawSchema", value:{$sum:"$value"} } },
105         { $out: `rawschemaordered${batchId}results` }
106         ]).allowDiskUse(true).exec().then((data) => {
107           return resolv(data);
108         }).catch((error) => {
109           console.error("error",error);
110           return reject(error);
111         });
112     });
113   }
114
115 }
```

Arquivo 133: codigofonte/server/controllers/rawSchema/rawSchemaUnorderedResult.ts

```
1  import * as jwt from 'jsonwebtoken';
2  import User from '../../models/user/user';
3  import PasswordHelper from '../../helpers/passwordHelper';
4  import BaseController from '../base';
5
6  export default class UserController extends BaseController {
7
8    model = User;
9
10   login = (email:String, password:String): Promise<any> => {
11     return new Promise((resolv, reject) => {
12       this.model.findOne({ 'email': email }).then((user) => {
13         if(!user)
14           return reject({"message":"Não foi possível encontrar sua
                 conta.", "code":404});
15         return user.comparePassword(password).then((isMatch) => {
16           if(!isMatch)
17             return reject({"message":"Senha incorreta. Tente
                   novamente.", "code":403});
18           return resolv({ 'token': jwt.sign({ 'user': user },
                 process.env.SECRET_TOKEN) });
19         });
20       });
21     });
22   }
23
24   getUser = (userId:String): Promise<any> => {
25     return new Promise((resolv, reject) => {
26       this.model.findOne({'_id': userId }).then((user) => {
27         if(!user)
28           return reject({"message":"Não foi possível encontrar sua
                 conta.", "code":404});
29         return resolv(user);
30       });
31     });
32   }
33
34   register = (req, res) => {
```

```
35      PasswordHelper.crypt(req.body.password).then((password) => {
36        req.body.password = password;
37        return this.insert(req, res);
38      }, (error) => {
39        return this.error(res, error, 500);
40      });
41    }
42  }
```

Arquivo 134: codigofonte/server/controllers/user/user.ts

```
1   class JSONSchema {
2
3     'definitions':any;
4     '$schema':String;
5     'id':String;
6     'title':String;
7     'type':String;
8     'properties':any;
9     'additionalProperties':Boolean;
10    'required':Array<String>;
11
12    constructor(){
13      this.$schema = "http://json-schema.org/draft-06/schema#";
14      this.definitions = {
15          "Binary":{
16            "title": "Binary",
17            "type": "object",
18            "properties": {
19                "$binary": {
20                    "type": "string"
21                },
22                "$type": {
23                    "type": "string"
24                }
25          },
26            "required": ["$binary", "$type"]
27          },
28          "Undefined":{
29            "title": "Undefined",
30            "type": "object",
31            "properties": {
32                "$undefined": {
33                    "type": "boolean"
34                }
35          },
36            "required": ["$undefined"]
37          },
38        "ObjectID": {
39          "title": "ObjectID",
40            "type": "object",
41            "properties": {
42                "$oid": {
43                    "type": "string"
```

```
            }
        },
          "required": ["$oid"]
        },
        "Date":{
          "title": "Date",
          "type": "object",
          "properties": {
              "$date": {
                  "type": "string"
              }
        },
          "required": ["$date"]
        },
        "RegExp":{
          "title": "RegExp",
          "type": "object",
          "properties": {
              "$options": {
                  "type": "string"
              },
              "$regex": {
                  "$type": "string"
              }
        },
          "required": ["$options", "$regex"]
        },
        "DBRef":{
          "title": "DBRef",
          "type": "object",
          "properties": {
              "$id": {
                  "type": "string"
              },
              "$ref": {
                  "type": "string"
              }
        },
          "required": ["$id", "$ref"]
        },
        "Code":{
          "title": "Code",
          "type": "object",
          "properties": {
              "$code": {
                  "type": "string"
              },
              "$scope":{
                "type": "object"
              }
        },
          "required": ["$code"]
        },
        "Timestamp":{
```

```
            "title": "Timestamp",
            "type": "object",
            "properties": {
                "$timestamp": {
                "type": "object",
                "properties": {
                  "i": {
                    "type":"integer"
                  },
                  "t": {
                    "type":"integer"
                  }
                },
                "required":["i","t"]
                }
            },
              "required":["$timestamp"]
          },
          "Long":{
            "title": "NumberLong",
            "type": "object",
            "properties": {
                "$numberLong": {
                    "type": "string"
                }
          },
            "required": ["$numberLong"]
          },
          "MinKey":{
            "title": "MinKey",
            "type": "object",
            "properties": {
                "$minKey": {
                    "type": "integer"
                }
          },
            "required": ["$minKey"]
          },
          "MaxKey":{
            "title": "MaxKey",
            "type": "object",
            "properties": {
                "$maxKey": {
                    "type": "integer"
                }
          },
            "required": ["$maxKey"]
          }
      };
      this.properties = {}
      this.additionalProperties = false;
    }
  }
  export default JSONSchema;
```

Arquivo 135: codigofonte/server/helpers/jsonSchema/jsonSchema.ts

```
import JSONSchema from './jsonSchema';
class JsonSchemaBuilder {
  rootSchema = new JSONSchema();
  usedDefinitions = [];
  build = (fields, count) => {
    this.rootSchema.properties = this.getFieldsSchema(fields);
    this.rootSchema.required = [];
    Object.keys(this.rootSchema.properties).forEach((property) => {
      if(this.rootSchema.properties[property].count === count)
        this.rootSchema.required.push(property);
      delete this.rootSchema.properties[property].count;
    });
    this.removeUnusedDefinitions();
    return this.rootSchema;
  }
  private getFieldsSchema = (fields) => {
    const properties = {};
    fields.forEach((field) => {
      properties[field.name] = this.getSchemaFromValue(field);
    });
    return properties;
  }
  private getSchemaFromValue = (value) => {
    let schema;
    if(value.types){
      schema = this.getFieldSchema(value);
    } else {
      if(this.isBSON(value)){
        schema = this.getExtendedTypeSchema(value);
      } else if(this.isPrimitive(value)){
        schema = this.getPrimitiveTypeSchema(null, value.name,
            value.count);
      } else if(value.name === "Array"){
        schema = this.getArrayTypeSchema(value);
      } else {
        schema = this.getObjectTypeSchema(value);
      }
    }
    return schema;
  }
  private getFieldSchema = (field) => {
    let schema;
    if(field.types.length === 1){
      const type = field.types[0];
      if(this.isBSON(type)){
        schema = this.getExtendedTypeSchema(type);
      } else if(this.isPrimitive(type)){
        schema = this.getPrimitiveTypeSchema(field.name, type.name,
            field.count);
      } else if(type.name === "Array"){
```

```
49          schema = this.getSchemaFromValue(type);
50        } else {
51          schema = this.getSchemaFromValue(type);
52          schema.name = type.path;
53        }
54      } else {
55        schema = {
56          "name": field.name,
57          "anyOf": this.buildItems(field.types),
58          "count": field.count
59        }
60        schema.anyOf.forEach((item) => {
61          delete item.count;
62        });
63      }
64      return schema;
65    }
66    private getExtendedTypeSchema = (type) => {
67      this.addToUsedDefinitions(type.name);
68      const schema = {
69        "$ref": `#/definitions/${type.name}`,
70        "count": type.count
71      };
72      return schema;
73    }
74    private getPrimitiveTypeSchema = (name, type, count) => {
75      let schema;
76      if(name){
77        schema = {
78          "name":name,
79          "type": type.toLowerCase(),
80          "count": count
81        }
82      } else {
83        schema = {
84          "type": type.toLowerCase(),
85          "count": count
86        }
87      }
88      return schema;
89    }
90    private getObjectTypeSchema = (object) => {
91      const schema = {
92        "type": "object",
93        "properties": this.getFieldsSchema(object.fields),
94        "count": object.count,
95        "additionalProperties": false,
96        "required":[]
97      }
98      Object.keys(schema.properties).forEach((property) => {
99        if(schema.properties[property].count === object.count){
100         schema.required.push(property);
101       }
102       delete schema.properties[property].count;
```

```
103       });
104       return schema;
105     }
106     private getArrayTypeSchema = (array) => {
107       let items, totalCount;
108       const itemsArray = this.buildItems(array.items);
109       if(itemsArray.length === 1){
110         items = itemsArray.pop();
111         totalCount = items.count;
112         delete items.count;
113       } else {
114         items = {
115           "anyOf": itemsArray
116         }
117         totalCount = itemsArray.map((item) => item.count).reduce((preVal
              , elem) => preVal + elem, 0);
118         itemsArray.forEach((item) => {
119           delete item.count;
120         });
121       }
122       const schema = {
123         "name": array.path,
124         "type": array.name.toLowerCase(),
125         "items": items,
126         "minItems":0,
127         "count": array.count,
128         "additionalItems":true
129       }
130       if(totalCount >= array.count)
131         schema.minItems = 1;
132       return schema;
133     }
134     private buildItems = (values) => {
135       const items = [];
136       values.forEach((value) => {
137         items.push(this.getSchemaFromValue(value));
138       });
139       return items;
140     }
141     private addToUsedDefinitions = (bsonType) => {
142       if (this.usedDefinitions.indexOf(bsonType) < 0)
143         this.usedDefinitions.push(bsonType);
144     }
145     private isBSON = (value) => {
146       switch(value.name){
147         case "Boolean":
148         case "Null":
149         case "String":
150         case "Number":
151         case "Array":
152         case "Object":
153         case "Integer":
154           return false;
155         default:
```

```
156           return true;
157       }
158     }
159     private isPrimitive = (value) => {
160       switch(value.name){
161         case "Boolean":
162         case "Null":
163         case "String":
164         case "Number":
165         case "Integer":
166           return true;
167         default:
168           return false;
169       }
170     }
171     private removeUnusedDefinitions = () => {
172       Object.keys(this.rootSchema.definitions).forEach((definition) => {
173         const definitionUsed = this.usedDefinitions.find((bsonType) => {
174           return bsonType === definition;
175         }) != null;
176         if(!definitionUsed)
177           delete this.rootSchema.definitions[definition];
178       });
179     }
180 }
181 export default JsonSchemaBuilder;
```

Arquivo 136: codigofonte/server/helpers/jsonSchema/jsonSchemaBuilder.ts

```
1  class ObjectKeysSorter {
2    sort(object){
3      let ordered;
4      if(typeof object === "string"){
5        ordered = object;
6      } else if (Array.isArray(object)){
7        const items = [];
8        object.forEach((item) => {
9          items.push(this.sort(item));
10       });
11       ordered = items.sort();
12     } else {
13       ordered = this.sortObject(object);
14       Object.keys(ordered).forEach((key) => {
15         if(ordered[key] !== undefined && ordered[key] !== null){
16           if(Array.isArray(ordered[key])){
17             const items = [];
18             ordered[key].forEach((item) => {
19               items.push(this.sort(item));
20             });
21             ordered[key] = items.sort();
22           } else if(typeof ordered[key] === 'object') {
23             ordered[key] = this.sort(ordered[key]);
24           }
25         }
```

```
      });
    }
    return ordered;
  }
  sortObject(o) {
      return Object.keys(o).sort().reduce((r, k) => (r[k] = o[k], r),
          {});
  }
}
export default ObjectKeysSorter;
```

Arquivo 137: codigofonte/server/helpers/objectKeysSorter.ts

```
import * as bcrypt from 'bcryptjs';
abstract class PasswordHelper {
  static crypt(password): Promise<any>{
    return new Promise((resolv, reject) => {
      bcrypt.genSalt(10).then((salt) => {
        bcrypt.hash(password, salt).then((hash) => {
          resolv(hash);
        }, (error) => {
          reject(error);
        });
        }, (error) => {
          reject(error);
        });
    });
  }
}
export default PasswordHelper;
```

Arquivo 138: codigofonte/server/helpers/passwordHelper.ts

```
abstract class BSONTypeHelper {
  static getBSONType(data) {
    if(data != null && data != undefined){
      if(data._bsontype != null){
        return data._bsontype;
      } else if (data.constructor.name === "Date"){
        return data.constructor.name;
      } else if (data.constructor.name === "RegExp"){
        return data.constructor.name;
      }
    }
  }
}
export default BSONTypeHelper;
```

Arquivo 139: codigofonte/server/helpers/rawSchema/bsonTypeHelper.ts

```
import {EventEmitter} from 'events';
import {ObjectID} from 'mongodb';
import RawSchemaProcessWorker from './rawSchemaProcessWorker';
```

```
4
5  class Executor {
6
7    execute = (collection, rawSchemaBatch): Promise<any> => {
8        return new Promise((resolv, reject) => {
9        const work = new RawSchemaProcessWorker().work(collection,
             rawSchemaBatch);
10           work.on('finalized', (data) => {
11             return resolv(data);
12           });
13           work.on('error', (error) => {
14         return reject({"type":"LOADING_DOCUMENTS_ERROR", "message":
               error.message, "code":400});
15           });
16       });
17   }
18
19 }
20 export default Executor;
```

Arquivo 140: codigofonte/server/helpers/rawSchema/executor.ts

```
1  import BSONTypeHelper from './bsonTypeHelper';
2  abstract class RawSchemaBuilder {
3    static build = (value) => {
4      const bsonType = BSONTypeHelper.getBSONType(value);
5      let instance;
6      if(bsonType){
7        instance = bsonType;
8      } else if (value === undefined){
9        instance = "Undefined";
10     } else if(value === null){
11       instance = "Null";
12     } else if (Array.isArray(value)){
13       instance = [];
14       value.forEach((arrayItem) => {
15         const arrayItemRawSchema = RawSchemaBuilder.build(arrayItem);
16         const rawSchemaAlreadyExists = instance.find((resp) => {
17           return JSON.stringify(resp) === JSON.stringify(
                 arrayItemRawSchema);
18         }) != null;
19         if(!rawSchemaAlreadyExists)
20           instance.push(arrayItemRawSchema);
21       });
22     } else if (typeof value === "object") {
23       instance = {};
24       Object.keys(value).forEach((property) => {
25         instance[property] = RawSchemaBuilder.build(value[property]);
26       });
27     } else {
28       instance = value.constructor.name;
29     }
30     return instance;
31   };
```

```
}
export default RawSchemaBuilder;
```

Arquivo 141: codigofonte/server/helpers/rawSchema/rawSchemaBuilder.ts

```
import {EventEmitter} from 'events';
import rawSchemaParser from './rawSchemaParser';
class RawSchemaDiscoverer extends EventEmitter {
  discovery(collection, batchId): EventEmitter {
    let result;
    const parser = collection.stream().pipe(rawSchemaParser(batchId));
    parser.on('data', (data) => { result = data; });
    parser.on('end', () => { this.emit('end',result); });
    parser.on('error',(error) => { this.emit('error',error); });
    return this;
  }
}
export default RawSchemaDiscoverer;
```

Arquivo 142: codigofonte/server/helpers/rawSchema/rawSchemaDiscoverer.ts

```
import * as es          from 'event-stream';
import ObjectKeysSorter from '../objectKeysSorter';
let order = function(){
  const mapper = es.through(function write(document) {
    try {
      let unorderedObject = JSON.parse(document.docRawSchema);
      let orderedObject = new ObjectKeysSorter().sort(unorderedObject)
          ;
      let orderedObjectJson = JSON.stringify(orderedObject);
      if(document.docRawSchema === orderedObjectJson){
        console.log("ficou igual");
        this.emit('ignore');
      } else {
        this.emit('progress', {'_id': document._id, 'docRawSchema':
            orderedObjectJson, 'batchId': document.batchId, 'value':
            document.value});
      }
    } catch (error) {
      console.error("grave error >>> ",error);
      console.error("document error >>> ",document);
    }

  }, function end() {
    this.emit('end');
  });
  mapper.on('close', function() {
    this.destroy();
  });
  return mapper;
}
export default order;
```

Arquivo 143: codigofonte/server/helpers/rawSchema/rawSchemaOrder.ts

```
import * as es          from 'event-stream';
import RawSchemaBuilder from './rawSchemaBuilder';
let parse = function(batchId: string){
  const rawSchemes = [];
  const mapper = es.through(function write(document) {
    const documentRawSchema = {};
    Object.keys(document).forEach((key) => {
      documentRawSchema[key] = RawSchemaBuilder.build(document[key]);
    });
    rawSchemes.push({
      "docId":document._id,
      "docRawSchema":JSON.stringify(documentRawSchema),
      "batchId":batchId
    });
    this.emit('progress', document);
  }, function end() {
    this.emit('data', rawSchemes);
    this.emit('end');
  });
  mapper.on('close', function() {
    this.destroy();
  });
  return mapper;
}
export default parse;
```

Arquivo 144: codigofonte/server/helpers/rawSchema/rawSchemaParser.ts

```
import {EventEmitter}     from 'events';
import RawSchemaDiscoverer  from './rawSchemaDiscoverer';

import RawSchemaController from '../../controllers/rawSchema/rawSchema
    ';

import {ObjectID} from 'mongodb';

export default class RawSchemaProcessWorker extends EventEmitter {

  limit = 5000;
  totalImported = 0;

  work = (collection, rawSchemaBatch): EventEmitter => {
    const saver = new RawSchemaController(rawSchemaBatch._id);
    let working = this.workNow(rawSchemaBatch, collection, null);
    working.on('save', (data) => {
      saver.saveAll(data, rawSchemaBatch._id).then((data) => {
        this.totalImported += data.length;
        console.log(this.totalImported," de ",
            rawSchemaBatch.collectionCount);
        if(this.totalImported >= rawSchemaBatch.collectionCount){
          rawSchemaBatch.status = "REDUCE_DOCUMENTS";
          rawSchemaBatch.extractionDate = new Date();
          rawSchemaBatch.save().then((data) => {
            this.emit('finalized', rawSchemaBatch);
```

```
25            });
26          }
27        }).catch((error) => {
28          this.emit('error', error);
29        });
30      })
31      .on('lastObjectId', (lastObjectId) => {
32        console.log("work in greater than ",lastObjectId);
33        working = this.workNow(rawSchemaBatch, collection, lastObjectId)
              ;
34      })
35      .on('error', (error) => {
36        this.emit('error', error);
37      });
38      return this;
39    }
40
41    workNow = (rawSchemaBatch, collection, lastObjectId): EventEmitter
        => {
42      const collectionToWork = collection.find(lastObjectId != null ? {'
          _id':{$gt:lastObjectId}} : {}).sort({_id:1}).limit(this.limit)
          ;
43      const discovery = new RawSchemaDiscoverer().discovery(
          collectionToWork, rawSchemaBatch._id);
44      discovery.on('end', (rawSchemes) => {
45        this.emit('save',rawSchemes);
46        if(rawSchemes.length > 0){
47          this.emit('lastObjectId',rawSchemes[rawSchemes.length-1].docId
              );
48        } else {
49          this.emit('done');
50        }
51      });
52      discovery.on('error',(error) => {
53        this.emit('error',error);
54      });
55      return this;
56    }
57  }
```

Arquivo 145: codigofonte/server/helpers/rawSchema/rawSchemaProcessWorker.ts

```
1   class RawSchemaUnifier {
2     private rootSchema = {
3       fields: [],
4       count: 0
5     };
6     union = (documents) => {
7       documents.forEach((document) => {
8         this.buildRawSchema(JSON.parse(document._id), Number(
              document.value), this.rootSchema.fields, null);
9         this.rootSchema.count = this.sum(this.rootSchema.count,
              document.value);
10      });
```

```
11      return this.rootSchema;
12    };
13    private buildRawSchema = (document, count, fields, parent) => {
14      Object.keys(document).forEach((key) => {
15        const path = parent != null ? `${parent}.JSONSCHEMADISCOVERY.${
              key}` : key;
16        this.addToField(path, document[key], fields, count);
17      });
18    };
19    private addToField = (specialPath, value, fields, count) => {
20      const lastPath = specialPath.split(".JSONSCHEMADISCOVERY.").pop();
21      const regexp = new RegExp(".JSONSCHEMADISCOVERY.", 'g');
22      const path = specialPath.replace(regexp, ".");
23      let field = fields.find((field) => {
24        return field.path === path;
25      });
26      field = this.getOrUpdateInstance(field, fields, lastPath, path,
            count);
27      if(!field.types)
28        field.types = [];
29      this.addToType(path, value, field.types, count);
30    };
31    private addToType = (path, value, types, count) => {
32      const typeName = this.getTypeFromValue(value);
33      let type = types.find((currentType) => {
34        return currentType.name === typeName;
35      });
36      type = this.getOrUpdateInstance(type, types, typeName, path, count
            );
37      this.workOnType(path, value, type, typeName, count);
38    };
39    private addToItem = (path, value, items, count) => {
40      const typeName = this.getTypeFromValue(value);
41      let item = items.find((currentItem) => {
42        return currentItem.name === typeName && currentItem.path ===
              path;
43      });
44      item = this.getOrUpdateInstance(item, items, typeName, path, count
            );
45      this.workOnType(path, value, item, typeName, count);
46    }
47    private workOnType = (path, value, instance, typeName, count) => {
48      if(typeName === 'Array'){
49        if(!instance.items)
50          instance.items = [];
51        value.forEach((arrayItem) => {
52          this.addToItem(path, arrayItem, instance.items, count);
53        });
54      } else if(typeName === 'Object'){
55        if(!instance.fields)
56          instance.fields = [];
57        this.buildRawSchema(value, count, instance.fields, path);
58      }
59    }
```

```
60    private getTypeFromValue = (value) => {
61      if(typeof value === "string"){
62        return value;
63      } else if (Array.isArray(value)){
64        return "Array";
65      } else {
66        return "Object";
67      }
68    };
69    private sum = (value1, value2) => Number(value1) + Number(value2);
70    private getOrUpdateInstance = (instance, instances, name, path,
          count) => {
71      if(!instance){
72        instance = {
73          "name":name,
74          "path":path,
75          "count":Number(count)
76        };
77        instances.push(instance);
78      } else {
79        instance.count = this.sum(instance.count, count);
80      }
81      return instance;
82    }
83  }
84  export default RawSchemaUnifier;
```

Arquivo 146: codigofonte/server/helpers/rawSchema/rawSchemaUnifier.ts

```
1  import * as mongoose from 'mongoose';
2  const AlertSchema = new mongoose.Schema({
3    'batchId': { type: mongoose.Schema.Types.ObjectId, ref: '
         RawSchemaBatch', required:true },
4    'userId': { type: mongoose.Schema.Types.ObjectId, ref: 'User',
         required:true },
5    'status': { type: String, required:true },
6    'type': { type: String, required:true },
7    'dbUri': { type: String, required:true },
8    'collectionName': { type: String, required:true },
9    'date': { type: Date, required:true },
10 },{ timestamps: { createdAt: 'createdAt' } });
11 export default mongoose.model('Alert', AlertSchema);
```

Arquivo 147: codigofonte/server/models/alert/alert.ts

```
1  import * as mongoose from 'mongoose';
2  const JsonSchemaExtractedSchema = new mongoose.Schema({
3    'batchId': { type: mongoose.Schema.Types.ObjectId, ref: '
         RawSchemaBatch', required:true },
4    'jsonSchema': { type: String, required:true }
5  },{ timestamps: { createdAt: 'createdAt' } });
6  JsonSchemaExtractedSchema.set('toJSON', {
7    transform: function(doc, ret, options) {
```

```
    ret.jsonSchema = JSON.parse(ret.jsonSchema);
    return ret;
  }
});
export default mongoose.model('JsonSchemaExtracted',
    JsonSchemaExtractedSchema);
```

Arquivo 148: codigofonte/server/models/jsonSchema/jsonSchemaExtracted.ts

```
import * as mongoose from 'mongoose';
import {Schema} from 'mongoose';
const rawSchemaSchema = new mongoose.Schema({
  'batchId': {type: mongoose.Schema.Types.ObjectId, ref: '
      RawSchemaBatch', required:true},
  'docRawSchema': { type: String, required: true},
  'docId': { type: String, required: true }
},{ timestamps: { createdAt: 'createdAt' } });
export default rawSchemaSchema;
```

Arquivo 149: codigofonte/server/models/rawSchema/rawSchema.ts

```
import * as mongoose from 'mongoose';

const rawSchemaBatchSchema = new mongoose.Schema({
  'userId': { type: mongoose.Schema.Types.ObjectId, ref: 'User',
      required:true },
  'collectionName': { type: String, required: true },
  'dbUri': { type: String, required: true },

  'collectionCount':{ type: Number, requried: true },
  'uniqueUnorderedCount':{ type: Number },
  'uniqueOrderedCount':{ type: Number },

  'status': { type: String, required: true },
  'statusMessage': { type: String },
  'statusType': { type: String },

  'reduceType': { type: String },

  'startDate': { type: Date },

  'extractionDate': { type: Date },
  'extractionElapsedTime': { type: Number },

  'unorderedMapReduceDate': {type:Date},
  'unorderedMapReduceElapsedTime': {type:Number},

  'orderedMapReduceDate': {type:Date},
  'orderedMapReduceElapsedTime': {type:Number},

  'unorderedAggregationDate': {type:Date},
  'unorderedAggregationElapsedTime': {type:Number},

```

```
32    'orderedAggregationDate': {type:Date},
33    'orderedAggregationElapsedTime': {type:Number},
34
35    'unionDate': {type:Date},
36    'unionElapsedTime': {type:Number},
37
38    'endDate': { type: Date },
39    'totalElapsedTime': { type: Number }
40
41  },{ timestamps: { createdAt: 'createdAt' } });
42
43  export default mongoose.model('RawSchemaBatch', rawSchemaBatchSchema);
```

Arquivo 150: codigofonte/server/models/rawSchema/rawSchemaBatch.ts

```
1  import * as mongoose from 'mongoose';
2  import {Schema} from 'mongoose';
3  const rawSchemaOrderedResultSchema = new mongoose.Schema({
4    'batchId': {type: mongoose.Schema.Types.ObjectId, ref: '
         RawSchemaBatch', required:false},
5    'docRawSchema': { type: String, required: false},
6    'value': { type: Number, required: false },
7    '_id': { type: Schema.Types.Mixed, required: false}
8  },{ timestamps: { createdAt: 'createdAt' } });
9  export default rawSchemaOrderedResultSchema;
```

Arquivo 151: codigofonte/server/models/rawSchema/rawSchemaOrderedResult.ts

```
1  import * as mongoose from 'mongoose';
2  const rawSchemaUnionSchema = new mongoose.Schema({
3    'batchId': {type: mongoose.Schema.Types.ObjectId, ref: '
         RawSchemaBatch', required:true},
4    'rawSchemaFinal': { type: String, required: true }
5  },{ timestamps: { createdAt: 'createdAt' } });
6  rawSchemaUnionSchema.set('toJSON', {
7    transform: function(doc, ret, options) {
8      ret.rawSchemaFinal = JSON.parse(ret.rawSchemaFinal);
9      return ret;
10   }
11 });
12 export default mongoose.model('RawSchemaUnion', rawSchemaUnionSchema);
```

Arquivo 152: codigofonte/server/models/rawSchema/rawSchemaUnion.ts

```
1  import * as mongoose from 'mongoose';
2  import {Schema} from 'mongoose';
3  const rawSchemaUnorderedResultSchema = new mongoose.Schema({
4    'batchId': {type: mongoose.Schema.Types.ObjectId, ref: '
         RawSchemaBatch', required:false},
5    'docRawSchema': { type: String, required: false},
6    'value': { type: Number, required: false },
7    '_id': { type: Schema.Types.Mixed, required: false}
8  },{ timestamps: { createdAt: 'createdAt' } });
```

```
9  export default rawSchemaUnorderedResultSchema;
```

Arquivo 153: codigofonte/server/models/rawSchema/rawSchemaUnorderedResult.ts

```
1  import * as bcrypt from 'bcryptjs';
2  import * as mongoose from 'mongoose';
3
4  const userSchema = new mongoose.Schema({
5    'username': { type: String, required: true },
6    'email': { type: String, unique: true, lowercase: true, trim: true,
         required: true },
7    'password': { type: String, required: true }
8  },{ timestamps: { createdAt: 'createdAt' } });
9
10 userSchema.methods.comparePassword = function(candidatePassword) {
11   return bcrypt.compare(candidatePassword, this.password);
12 };
13
14 userSchema.set('toJSON', {
15   transform: function(doc, ret, options) {
16     delete ret.password;
17     return ret;
18   }
19 });
20
21 export default mongoose.model('User', userSchema);
```

Arquivo 154: codigofonte/server/models/user/user.ts

```
1  class AuthenticationParam {
2    authDatabase:String;
3    userName:String;
4    password:String;
5    authMechanism:String;
6
7    isValid = (): boolean => {
8      if(this.authDatabase && this.userName && this.password &&
           this.authMechanism)
9        return true;
10     return false;
11   }
12 }
13 export default AuthenticationParam;
```

Arquivo 155: codigofonte/server/params/authenticationParam.ts

```
1  import AuthenticationParam from './authenticationParam';
2  class DatabaseParam {
3    address:string;
4    port:string;
5    authentication:AuthenticationParam;
6    userId:string;
7    databaseName:string;
```

```
8     collectionName:string;
9     constructor(data) {
10      Object.assign(this, data);
11      this.address = this.address.replace(/(^\w+:|^)\/\//, '');
12    }
13    public getURI() {
14      let uri;
15      if(this.hasAuthentication()){
16        uri = this.getURIWithAuthentication(`${this.address}:${this.port
              }/${this.databaseName}`);
17      } else {
18        uri = this.getURIWithoutAuthentication();
19      }
20      return `mongodb://${uri}`;
21    }
22    public getURIWithoutAuthentication() {
23      return `${this.address}:${this.port}/${this.databaseName}`;
24    }
25    public getURIWithAuthentication(address) {
26      return `${this.authentication.userName}:${
            this.authentication.password}@${address}?authSource=${
            this.authentication.authDatabase}&authMechanism=${
            this.authentication.authMechanism}`;
27    }
28    private hasAuthentication(){
29      if(this.authentication && this.authentication.authDatabase &&
            this.authentication.userName && this.authentication.password
            && this.authentication.authMechanism)
30        return true;
31      return false;
32    }
33  }
34  export default DatabaseParam;
```

Arquivo 156: codigofonte/server/params/databaseParam.ts

```
1   let options = {
2     verbose: true,
3     map: null,
4     reduce: null,
5     scope: null,
6     finalize: null,
7     out: {},
8     query: {},
9     sort: null,
10    jsMode: false
11  };
12  export default options;
```

Arquivo 157: codigofonte/server/params/mapReduceParam.ts

```
1   import * as express from 'express';
2   import UserController from './controllers/user/user';
```

```
3  import RawSchemaBatchController from './controllers/rawSchema/
       rawSchemaBatch';
4  import JsonSchemaExtractedController from './controllers/jsonSchema/
       jsonSchemaExtracted'
5  import ApiController from "./api/api";
6
7  export default function setRoutes(app) {
8
9    const router = express.Router();
10
11   const apiController = new ApiController();
12
13   const userController = new UserController();
14   const rawSchemaBatchController = new RawSchemaBatchController();
15   const jsonSchemaExtractedController = new
         JsonSchemaExtractedController();
16
17   router.route('/login').post(apiController.login);
18   router.route('/register').post(userController.register);
19   router.route('/user').get(apiController.getUser);
20
21   router.route('/batch/:id').get(rawSchemaBatchController.get);
22   router.route('/batch/:id').delete(apiController.deleteBatch);
23   router.route('/batches').get(apiController.listBatchesByUserId);
24
25   router.route('/alert/:id').delete(apiController.deleteAlert);
26   router.route('/alerts').get(apiController.listAlertsByUserId);
27   router.route('/alerts/count').get(apiController.countAlertsByUserId)
         ;
28
29   router.route('/batch/rawschema/steps/all').post(
         apiController.allSteps);
30   router.route('/batch/rawschema/discovery').post(
         apiController.discovery);
31   router.route('/batch/rawschema/reduce').post(apiController.reduce);
32   router.route('/batch/rawschema/aggregate').post(
         apiController.aggregate);
33   router.route('/batch/rawschema/aggregateAndReduce').post(
         apiController.aggregateAndReduce);
34   router.route('/batch/rawschema/union').post(apiController.union);
35   router.route('/batch/jsonschema/generate').post(
         apiController.generate);
36   router.route('/batch/jsonschema/generate/:id').get(
         jsonSchemaExtractedController.listByBatchId);
37
38   // Apply the routes to our application with the prefix /api
39   app.use('/api', router);
40
41 }
```

Arquivo 158: codigofonte/server/routes.ts

```
1  {
2    "extends": "../tsconfig.json",
```

```
3     "compilerOptions": {
4       "outDir": "../dist/server",
5       "baseUrl": "",
6       "module": "commonjs"
7     }
8   }
```

Arquivo 159: codigofonte/server/tsconfig.json