Implementando Self Service Business Intelligence utilizando a técnica de *Scaffolding*[1]

Ruan Carlos Alves da Silva, Rodrigo de Almeida Vilar de Miranda

Departamento de Ciências Exatas (DCX) – Universidade Federal da Paraíba (UFPB)
Rio Tinto – PB – Brazil

{ ruan.carlos, rodrigovilar} @dcx.ufpb.br

Abstract. *The implementation of Business Intelligence (BI) has a high financial cost, as it requires specialized labor. However, in order to solve the problems mentioned above, the concept of Self Service Business Intelligence (SSBI) was created. SSBI consists of allowing casual users to access BI resources, through software that offers resources ready to be interpreted by users, without having to involve an IT department and BI specialists. In this work, an implementation of the SSBI concept was proposed with the Metabase tool and adapted as a module for Code Generators, based on the Scaffolding technique. The main results of this work are: adaptation of the Code Generation concepts to create personalized BI Resources according to the metadata, an implementation of the proposed solution and the analysis of the solution in relation to the SSBI challenges listed by the academic community.*

Resumo. *A implementação de Business Intelligence (BI) tem um alto custo financeiro, pois demanda mão de obra especializada. Todavia, com a finalidade de solucionar os problemas mencionados acima, foi criado o conceito de Self Service Business Intelligence (SSBI). SSBI consiste em permitir que os usuários casuais acessem os recursos de BI, através de um software que oferece recursos prontos para serem interpretados pelos usuários, sem precisar envolver um departamento de TI e especialistas em BI. Neste trabalho foi proposta uma implementação do conceito de SSBI com a ferramenta Metabase e adaptada como um módulo para Geradores de Código, baseando-se na técnica de Scaffolding. Os principais resultados deste trabalho são: adaptação dos conceitos de Geração de Código para criação de Recursos de BI personalizados de acordo com os metadados, uma implementação da solução proposta e a análise da solução em relação aos desafios de SSBI elencados pela comunidade acadêmica.*

[1] "Trabalho de conclusão de curso, sob orientação do professor Rodrigo de Almeida Vilar de Miranda submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO."

1. Introdução

Os sistemas transacionais tendem a acumular um grande volume de dados. Tendo em vista a importância das informações que são geradas a partir desses dados, as corporações buscam implementar ferramentas que combinam os dados acumulados para gerar recursos que auxiliam nas tomadas de decisões. Portanto é comum que as instituições tenham interesse em implementar *Business Intelligence* (BI). Segundo Braghittoni (2017), BI é um termo cunhado por Howard Dresner do Gartner Group, em 1989, para descrever um conjunto de conceitos e métodos para melhorar o processo de tomada de decisão das empresas, utilizando-se de sistemas fundamentados em fatos e dimensões sobre os dados.

Um dos grandes problemas da implementação do BI, principalmente em empresas de médio e pequeno porte, é o alto custo no desenvolvimento da solução. E o custo não é justificado apenas por conta das licenças de soluções pagas. Mesmo nas soluções código aberto ou gratuitas existem os custos de customização, configuração, estrutura de hardware, consultorias, treinamentos, suporte técnico, implementação e etc. Dessa forma, é possível observar uma oportunidade de atender as pequenas empresas em suas demandas de BI, desenvolvendo uma solução de baixo custo e fácil de usar.

Uma das soluções para essa problemática está na criação de uma nova ferramenta, produto da junção do conceito de *Self-Service BI* e a técnica de geração de código, na qual seja possível o próprio usuário final usufruir dos seus recursos de BI com facilidade, e sem se preocupar em construir uma estrutura complexa para isso. Segundo Alpar e Schulz (2016), SSBI é capacitar usuários casuais a realizar análises personalizadas e derivar informações acionáveis de grandes quantidades de dados multifacetados sem precisar envolver especialistas em BI.

O objetivo desse trabalho é projetar e validar uma arquitetura de software para uma solução de SSBI adaptada a partir da técnica de geração de código baseada em *Scaffolding* e integrada com a ferramenta de código aberto Metabase[2], onde o termo Scaffolding é empregado para programas que utilizam a geração automática de código-fonte baseado em templates, contudo de acordo com Franky e Pavlich-Mariscal (2012), esses templates são modelos de código de soluções bem sucedidas, podendo assim serem utilizados como padrões para desenvolvimento.

Os sistemas de geração de código utilizam metadados (nome das entidades, propriedades, tipos, regras de validação, descrição dos relacionamentos e etc) para gerar componentes visuais (listagens, formulários, recursos de BI) e serviços (validação, persistência). Segundo Niso (2004), metadados são informações que explicam, localizam ou facilitam a manipulação de uma fonte de informação. A principal proposta

[2] https://www.metabase.com/

neste trabalho é a análise dos metadados a fim de viabilizar que sejam utilizados para gerar, configurar e implantar recursos de BI para o sistema no momento em que o seu código está sendo gerado.

Os geradores de código baseados em *Scaffolding* existentes não possuem suporte para gerar recursos de BI. Esta afirmação é apoiada na leitura dos textos de Costa (2018), Verma (2014) e Sasidharan (2018). Portanto, na proposta desse trabalho, o gerador de código deve ser capaz de calcular os recursos de BI possíveis a partir dos metadados dos atributos de uma entidade simples. O tipo da propriedade e suas restrições de unicidade e obrigatoriedade serão utilizados para calcular e configurar automaticamente os relatórios e gráficos possíveis no Metabase, e cada usuário poderá escolher quais são relevantes para si, ocultando ou exibindo os recursos no sistema que foi gerado.

A estrutura do artigo está dividida nas seguintes Seções. A segunda Seção trata da fundamentação teórica sobre SSBI, Geração de Código baseada em *Scaffolding* e Metabase. A Seção 3 descreve a metodologia do trabalho. A quarta Seção aborda a adaptação dos conceitos de geração de código para SSBI, a fim de tornar o gerador de código capaz de calcular os recursos de BI que podem ser gerados automaticamente. A Seção 5 apresenta a arquitetura da solução proposta. A sexta Seção expõe a análise da solução em relação aos desafios de SSBI e a sétima Seção descreve as conclusões da pesquisa.

2. Fundamentação Teórica

Esta seção tem o objetivo de definir os elementos utilizados para propor a solução do problema destacado neste trabalho: *Self-Service BI*, Geração de Código baseada em *Scaffolding* e a ferramenta Metabase.

2.1. *Self Service BI*

Segundo Lennerholt et al. (2018), SSBI é permitir que todos os usuários acessem dados e conduzam suas próprias análises para a tomada de decisão, sem precisar envolver um departamento de TI e usuários especializados em BI. Segundo Negash e Gray (2008) BI é um sistema de suporte à decisão orientado a dados que combina coleta de dados, armazenamento e gerenciamento de conhecimento com análise para fornecer entrada para o processo de decisão. A grande diferença entre os conceitos de SSBI e BI é que no SSBI é possível o usuário consumir recursos de BI com mais facilidade, oferecidos previamente pelo próprio sistema, já com BI é necessário que um especialista desenvolva os recursos para fornecer ao usuário comum.

2.2. Geração de Código baseada em *Scaffolding*

Desenvolver um software com qualidade é uma tarefa que requer esforço e conhecimento. Além do mais, é necessário que a equipe de desenvolvimento disponha

de disciplina para seguir os padrões de construção de código estabelecidos. Segundo Pressman (2006) se essas dificuldades não forem sanadas podem resultar em baixa produtividade, baixa qualidade de software e insatisfação do cliente.

Portanto reutilizar padrões de código é uma solução interessante, e para isso existe o conceito de geração de código. Segundo Herrington (2003), a geração de código é uma técnica semelhante a os padrões de projetos, que consiste em escrever programa que escrevem outros programas, ou seja, os geradores de código são ferramentas de produção automatizada de artefatos que permitem o reuso de código e que contribuem para a redução de tempo e dos custos empregados no desenvolvimento de software.

Assim sendo, um gerador de código pode ser uma opção de ferramenta para automatizar a criação da aplicação que manipula os recursos de BI. Na prática o gerador pode construir a aplicação com base nos *templates* já registrados, dessa forma fornecendo recursos de BI já prontos sem a necessidade de configurá-los manualmente.

Contudo para alcançar o objetivo de exibir modelos prontos de recursos de BI, filtrados por tipos e restrições atribuídos aos dados, tem-se a necessidade de inserir essas informações no momento que for gerar o código da aplicação. Essa questão pode ser solucionada utilizando metadados. De acordo com Ikematu (2001), metadados são dados que descrevem atributos de um recurso. Ele suporta um número de funções: localização, descoberta, documentação, avaliação, seleção e etc.

Para realizar uma boa integração entre o conceito de SSBI e Geração de Código, existe uma técnica que se encaixaria muito bem com a proposta, que seria a técnica de *Scaffolding*. Conforme Adamus et al. (2010), a técnica foi popularizada pela plataforma *Ruby on Rails*[3] e permite que se implemente uma aplicação CRUD web funcional em minutos, simplesmente mapeando metadados de objetos, banco de dados e interfaces. Além do mais, existem outros geradores bem populares baseados em *Scaffolding,* são eles: *jHipster*[4] e *Yeoman*[5].

2.3. Metabase

Para implementar um sistema com SSBI é recomendável utilizar um serviço que forneça recursos de BI. Contudo criar uma ferramenta que disponha dessas funcionalidades demandaria muito tempo e recursos. No mercado existem poderosas ferramentas que oferecem esse tipo de serviço. No entanto para baratear os custos e facilitar o consumo de seus recursos a aplicação deve ser gratuita, escalável e possuir uma API REST.

[3] https://guides.rubyonrails.org/
[4] https://www.jhipster.tech/
[5] https://yeoman.io/

Foi proposta por Falcone e Vilar (2020) a ferramenta Metabase como uma solução, pois a análise dos autores aponta a aplicação como sendo a melhor opção atendendo os seguintes critérios: Possuir código aberto, ser gratuita, possuir arquitetura escalável, possuir poucas dependências, permitir integração na interface de um sistema existente, permitir integração na autenticação, possuir boa documentação, comunidade ativa, possui Interface web, possuir API REST e não haver limitações na conta gratuita. Tendo em vista que os critérios também estão alinhados com as necessidades deste trabalho, a ferramenta Metabase foi selecionada para desempenhar a função de fornecer recursos de BI.

3. Metodologia

Para atingir o objetivo geral da pesquisa, que é propor um modelo de arquitetura, com o intuito de facilitar a utilização de BI por usuários comuns, foi necessário realizar três atividades. A primeira parte foi a investigação, onde foram realizados os estudos sobre Geração de Código baseada em *Scaffolding*, SSBI e a ferramenta Metabase. A segunda parte foi o projeto da solução, que é a integração de SSBI em Geradores de Código. A terceira parte foi a validação da solução proposta em relação aos desafios de SSBI já catalogados na literatura acadêmica.

3.1. Investigação

Esse procedimento iniciou-se com a pesquisa sobre o conceito de SSBI, tendo em vista que é a concepção no qual está sendo apoiado o contexto do trabalho. A ferramenta utilizada como motor de busca foi o *Google Scholar*. A string de busca utilizada foi "Self-Service BI", e a partir dela foram encontrados aproximadamente 32.000 resultados. Dessa lista foram escolhidos os nove trabalhos mais relevantes, e com base na leitura do resumo de cada um deles foram selecionados dois, levando em consideração o critério de alinhamento com o objetivo deste trabalho. Os dois documentos são de Lennerholt et al. (2018) e Alpar e Schulz (2016).

Logo após, a pesquisa foi direcionada para os geradores de código baseados em *Scaffolding* e os metadados, com o intuito de encontrar uma solução para a integração do Metabase a um gerador de código. A investigação sobre os geradores ocorreu no livro "Code generation in action" de Herrington (2003), e o conteúdo sobre técnica de *Scaffolding* foi encontrado no trabalho de Adamus et al. (2010) localizado nas referências do artigo de Falcone e Vilar (2020). Sobre os metadados, a investigação foi realizada de duas formas, a primeira foi executada nos motores de busca da ferramenta *Google Scholar*, a string de busca utilizada foi "Metadado" e foram encontrados aproximadamente 61.000 resultados. Dos resultados foram escolhidos quatro de acordo com a posição de relevância estabelecida pela ferramenta de busca, e com base na leitura do resumo de cada documento foram selecionados dois, que atendiam aos

critérios de alinhamento com o contexto deste trabalho, os trabalhos escolhidos foram de Niso (2004) e Ikematu (2001).

Por fim iniciou-se a pesquisa sobre a ferramenta Metabase, através de motores de busca do Google e do Youtube. Contudo a principal fonte de estudos sobre a aplicação foi a sua própria documentação. Além disso uma página que também recebeu destaque em relação ao estudo da ferramenta foi o GitHub com diversas discussões no repositório sobre o sistema.

3.2. Projeto da Solução

Com o objetivo de analisar e definir a melhor forma de integrar o Metabase ao gerador de código, e assegurar a aplicação correta do conceito de SSBI, foram realizadas algumas provas de conceito (do inglês *Proof of Concept* - PoC). Dentre as PoCs realizadas, destacam-se:

- **Implementação dos recursos de BI no sistema gerado**, onde é demonstrado como foi modelado o banco de dados para receber os recursos de BI no sistema gerado, e como foi realizada a ação de incorporar os *Cards* e *Dashboards* do Metabase no sistema gerado;
- **Configuração de *Cards* a partir dos metadados do CRUD**, no qual foram criados JSONs de configuração, para padronizar o modelo do corpo da requisição que o gerador de código precisa utilizar para criar os *Cards* no Metabase;
- **Autenticação**, no qual foi necessário definir como seria a melhor forma de comunicação entre o sistema gerado e o Metabase para realizar o login do usuário nas duas aplicações;

3.3. Validação com base na literatura

Entre os trabalhos encontrados foi identificado o catálogo de dez desafios na revisão da literatura de Lennerholt et al. (2018), referente ao SSBI. Devido ao alinhamento com o objetivos do trabalho, os critérios foram escolhidos para validar as características da solução proposta como resultado.

4. Adaptação dos conceitos de Geração de Código para criação de Recursos de BI

A estrutura dos metadados foram estudadas nesta Seção, com a finalidade de atribuir ao gerador de código a responsabilidade de calcular e configurar automaticamente os recursos de BI que podem ser disponibilizados para o usuário em um sistema gerado. Inicialmente foram estudadas as restrições de integridade. Posteriormente foram analisadas as combinações possíveis com base nas restrições de integridade, e quais funções ou instruções do SQL cada combinação pode executar. A Seção finaliza com a

análise das possibilidades de geração de informação através de diagramas e com um exemplo para a validação da abordagem proposta.

4.1. Restrições de Integridade dos Dados

Segundo Coronel et al. (2010), os dados são fatos brutos, e a palavra bruto indica que os fatos ainda não foram processados para revelar seu significado. Pode-se dizer que os dados são pequenos fragmentos, que combinados geram informação. Mediante esse conceito, é possível afirmar que sem serem processados os dados são apenas partículas soltas sem um propósito definido.

As aplicações de banco de dados possuem restrições de integridade que devem complementar os dados. Segundo Date (2000), as restrições de integridade servem para fornecer a garantia de que mudanças feitas no banco de dados por usuários autorizados não resultem em perda da consistência dos dados, protegendo o banco de danos acidentais.

De acordo com Elmasri et al. (2005), o tipo mais simples de restrições de integridade envolve a especificação de um tipo de dado para cada item. Segundo Lazzaretti et al. (2005), as restrições de tipo são apenas uma declaração para o efeito de que um atributo seja de um tipo especificado, ou seja, as restrições de tipo fazem parte da definição do atributo em questão e podem ser identificadas por meio do tipo correspondente. Por exemplo, na definição do atributo VDATA (DATE), o atributo VDATA possui o seu valor limitado ao tipo DATE.

Além das restrições de tipo, também é considerável para o nosso contexto as restrições básicas de tabelas. Lazzaretti et al. (2005) descrevem que restrições básicas de tabela podem ser: uma definição de chave candidata (definida através das cláusulas PRIMARY KEY e UNIQUE), uma definição de chave estrangeira (definida através da cláusula FOREIGN KEY) e uma definição de restrição de verificação (definida através da cláusula CHECK).

4.1.1. Restrições de Tipo

As restrições de tipo utilizadas no trabalho são três: *Integer*, referente a números inteiros; *Text*, que serve para representar os dados do tipo texto; e *Real Number*, utilizado para classificar os dados que são números reais. Existem outras restrições de tipo, porém não foram inseridas no escopo deste trabalho.

4.1.2. Restrições Básicas de Tabela

As restrições básicas de tabela são importantes quando o objetivo é abstrair quais dados podem ser utilizados para gerar uma informação específica. Os tipos podem ser classificados em Unique (U), quando o valor do dado não pode ser repetido, Not Unique (!U), ao contrário do anterior o valor pode ser repetido, Mandatory (M), o valor do item

não pode ser nulo e Not Mandatory (!M), em oposição ao valor anterior o dado pode ser nulo.

4.2. Funções e Instruções

Para definir o domínio de possibilidade de informações geradas com base nos metadados fornecidos ao gerador de código, foi preciso realizar um estudo sobre métodos que são utilizados para gerar informações a partir de dados. Foi observado que o próprio SQL fornece diversas funções e instruções que compartilham desse objetivo.

Portanto foram analisadas cinco funções e instruções do SQL, são elas: Count, que tem o objetivo de contabilizar os itens não nulos de uma coluna; Max, que tem a função de selecionar o maior número em uma coluna com os tipos *numeric*, *integer* entre outros relacionados a contagem; Min, é semelhante à função anterior, com diferencial que ao invés de selecionar o maior número da coluna, irá selecionar o menor número; Avg, também é semelhante à função anterior, porém o seu diferencial é de retornar a média dos valores de uma coluna; *Group By*, é uma instrução que tem a finalidade de agrupar valores por uma coluna.

4.3. Definir a combinação dos Tipos de Valor

Nesta Subseção, são analisadas as combinações possíveis nas restrições dos metadados, a fim de definir que funções SQL podem ser chamadas para gerar *cards* de *dashboards* para cada coluna de uma tabela simples. A Tabela 1 representa a combinação entre as restrições básicas de tabela dos dados.

Tabela 1. Tabela das combinações de Restrições Básicas

	M (Obrigatório)	**!M (Opcional)**
U (Único)	Obrigatório Único	Opcional Único
!U (Repetível)	Obrigatório Repetível	Opcional Repetível

De acordo com a tabela acima, existem quatro combinações. Cada uma dessas combinações tem um conjunto com as possibilidades de funções ou instruções possíveis de executar, levando em consideração as restrições atribuídas a cada uma delas, ver Tabela 2.

Tabela 2. Tabela das funções SQL disponíveis para cada Combinação

Combinação	**Funções disponíveis**
Obrigatório Único	*Count*
Opcional Único	*Count Null* e *Count*

Obrigatório Repetível	*Count* e *Group By*
Opcional Repetível	*Count Null*, *Count* e *Group By*

4.4. Diagramas

Após a análise dos tipos dos dados e restrições, juntamente com as funções que poderiam ser aplicadas em cada campo das entidades, foram criados os diagramas das Figuras 1, 2 e 3 para agregar a combinações das restrições de tipo e documentar as possibilidades de geração de relatórios e gráficos com as funções disponíveis para cada caso.

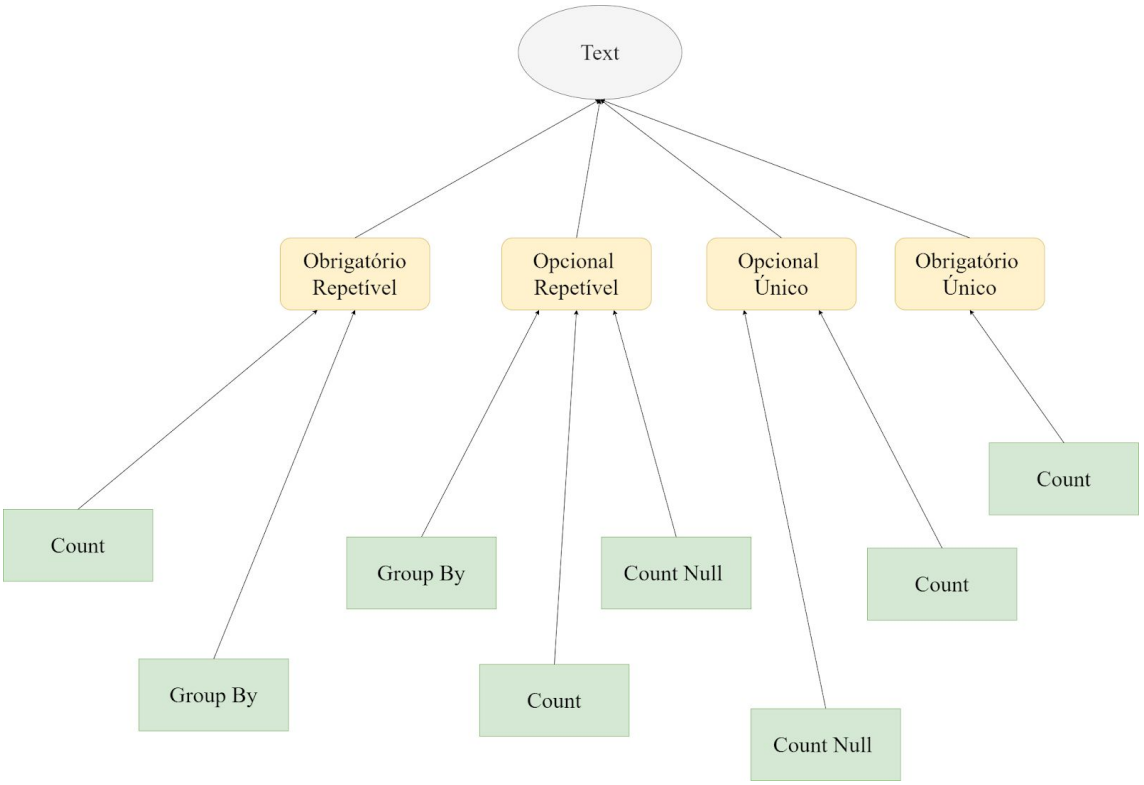

Figura 1. Possibilidades para o Tipo de Dado Text

A Figura 1 representa o diagrama com as possibilidades de geração de informações para a restrição de tipo *Text.* Através do diagrama é presumível observar que as possibilidades de gerar informações são definidas em três: *Count*, *Group By* e *Count Null*.

A Figura 2 representa o diagrama com as possibilidades de geração de informações para a restrição de tipo *Real Number.* Através do diagrama é presumível observar que as possibilidades de gerar informações são definidas em três: *Count*, *Histogram* e *Count Null*.

A Figura 3 representa o diagrama com as possibilidades de geração de informações para a restrição de tipo *Integer.* Através do diagrama é presumível observar que as possibilidades de gerar informações são definidas em sete: *Count*, *Histogram*, *Min*, *Max*, *Avg*, *Group By* e *Count Null*.

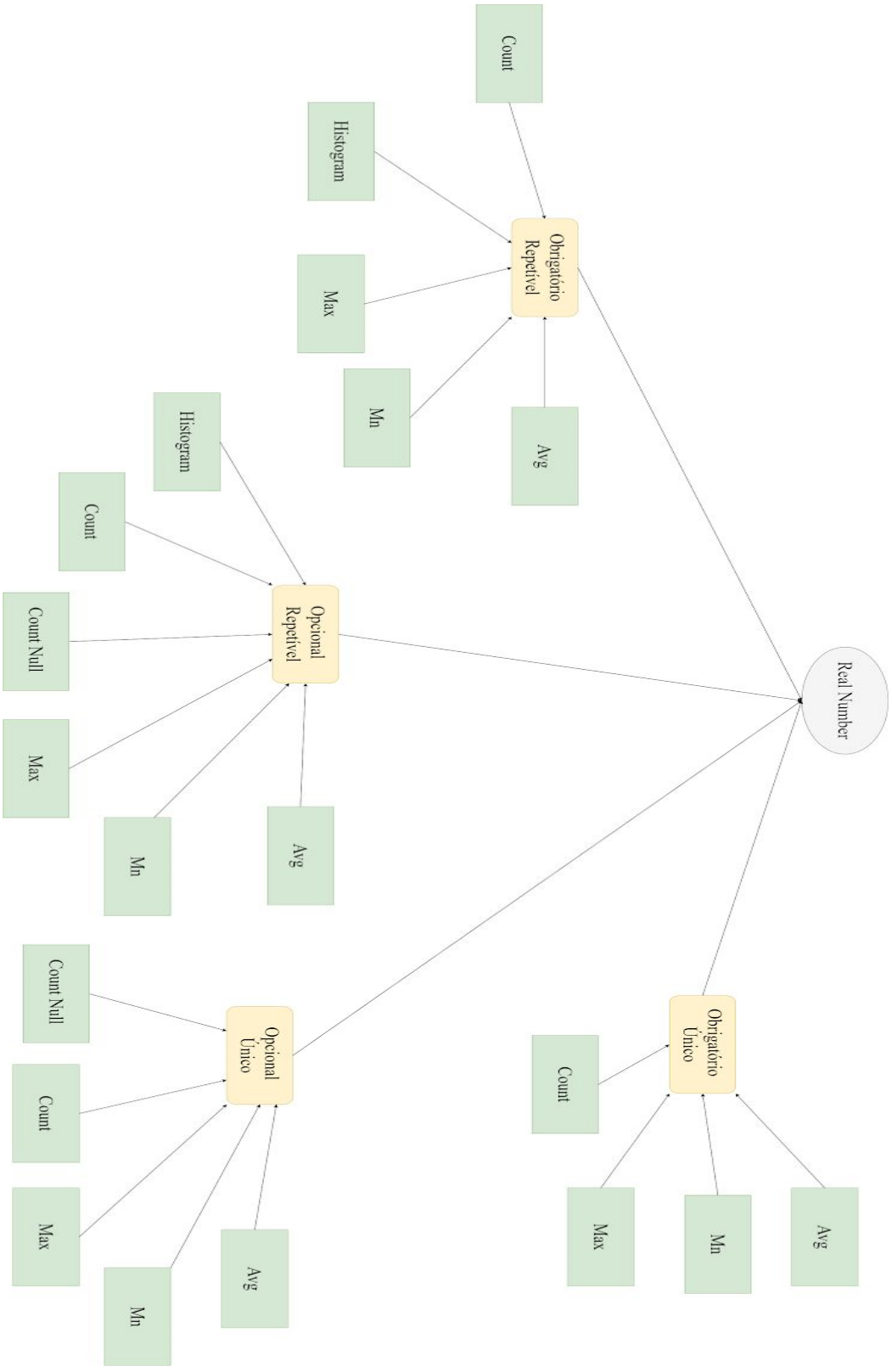

Figura 2. Possibilidades para o Tipo de Dado Real Number

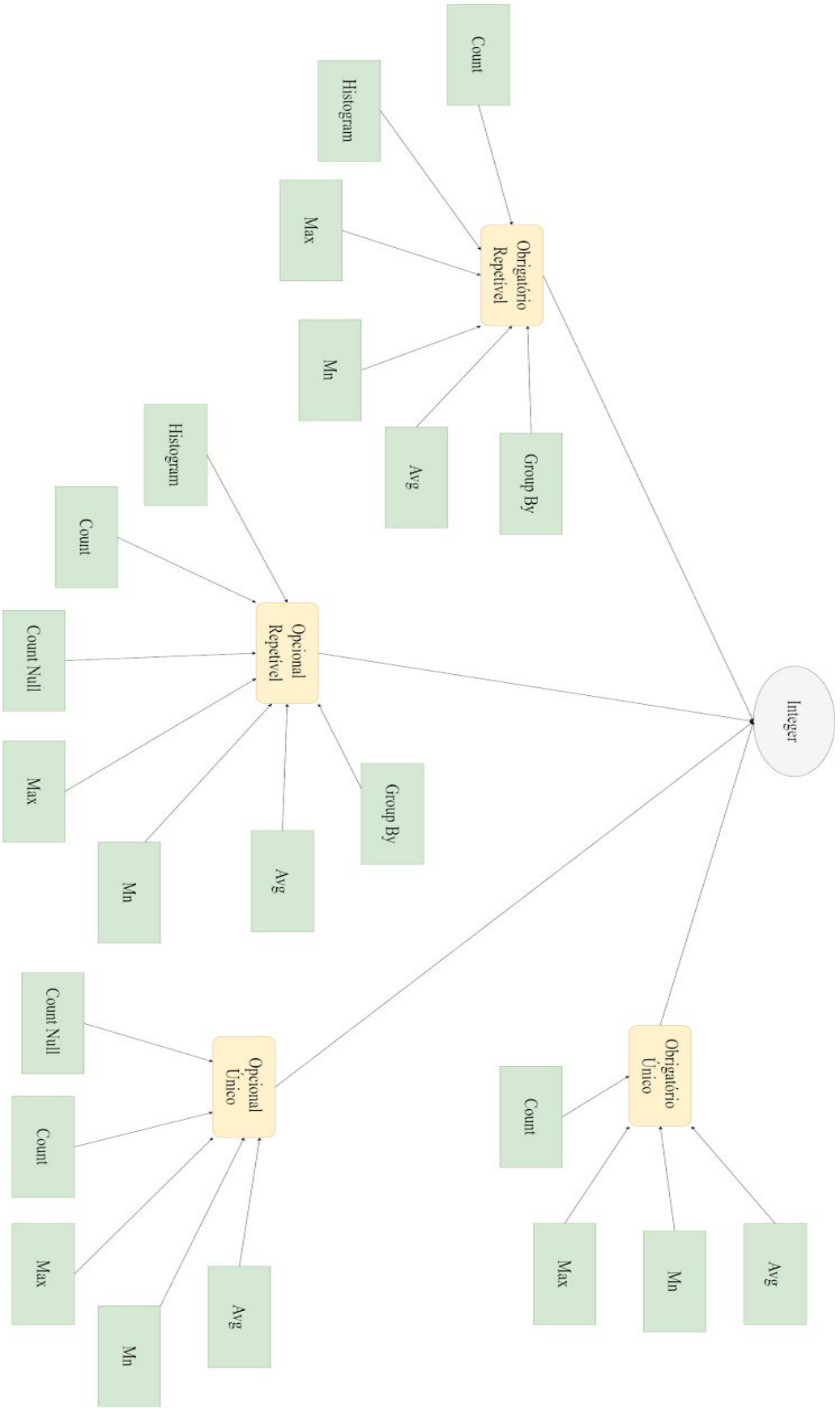

Figura 3. Possibilidades para o Tipo de Dado Integer

4.5. Validação

Para realizar a validação dos conceitos representados nas Tabelas 1 e 2 e nas Figuras 1, 2 e 3, foi utilizado um exemplo de tabela relacional (Tabela 3). A cada uma das colunas são atribuídas restrições de integridade de tipo e restrições básicas de tabela. E com esses detalhes de cada coluna foi possível descrever quais informações seria possível processar para cada uma das combinações. As combinações são: Obrigatório Único (OBU), Obrigatório Repetível (OBR), Opcional Único (OPU) e Opcional Repetível (OPR).

Tabela 3. Exemplo de Tabela relacional de Usuário com combinações de restrições

id **OBU** **Type: Integer**	***name*** **OBR** **Type: Text**	***last_name*** **OPR** **Type: Text**	***username*** **OBU** **Type: Text**	***telephone*** **OPU** **Type: Text**	***age*** **OBR** **Type: Integer**	***height*** **OPR** **Type: Double**
1	Lúcio	Carlos	lucarlos	96630456	38	1.80
2	Eduarda	Alves	edalves	97735666	21	1.65
3	Marcos	Carlos	mcarlos	97605140	29	1.72
4	Rivaldo	Silva	rsilva	99703800	32	
5	Eduarda	Carneiro	ecarneiro		19	1.70
6	Maria	Carneiro	mcarneiro		21	1.68
7	Antônio	Silva	asilva		29	
8	Marcos		mrcs		38	1.82
9	Maria	Silva	msilva	95884030	41	
10	Augusto		agst		29	1.75

A coluna **id** é do tipo **Integer e** possui a restrição OBU (Obrigatório Único). Na Figura 3, pode-se ver que para OBU as funções possíveis são: Count, que resultaria em um *Card* com o número 10. Max, que também resultaria em um *Card* com o número 10. Min, que resultaria em um *Card* com o número 1. E Avg, que resultaria em um card com o número 5,5.

A coluna ***name*** é do tipo **Text** e possui a restrição OBR (Obrigatório Repetível). Na Figura 1, é possível observar que para OBR as funções disponíveis são: Count, que

resultaria em um *Card* com o número 10. E *Group By* cujo resultado é exibido na Tabela 4.

Tabela 4. *Group By name*: exemplo de recurso de BI para coluna obrigatória repetível do tipo Text

name	**quantity**
Lúcio	1
Eduarda	2
Marcos	2
Rivaldo	1

A coluna **last_name** é do tipo **Text** e possui a restrição OPR (Opcional Repetível). Na Figura 1, é possível observar que para OPR as funções disponíveis são: *Count*, que resultaria em um *Card* com o número 8. *Count Null,* que resultaria em um *Card* com o número 2. E *Group By* cujo resultado é mostrado na Tabela 5.

Tabela 5. *Group By* last_name: exemplo de recurso de BI para coluna opcional repetível do tipo Text

last_name	**quantity**
Carlos	2
Alves	1
Silva	3
Carneiro	2

A coluna **username** é do tipo **Text** e possui a restrição OBU (Obrigatório Único). Na Figura 1, é possível observar que para OBU a única função possível é Count, que resultaria em um *Card* com o número 10.

A coluna **telephone** é do tipo **Text** e possui a restrição OPU (Opcional Único). Na Figura 1, é possível observar que para OPU as funções disponíveis são: *Count*, que resultaria em um *Card* com o número 5. E *Count Null,* que resultaria em um *Card* também com o número 5.

A coluna **age** é do tipo **Integer** e possui a restrição OBR (Obrigatório Repetível). Na Figura 3, é possível reparar que para OBR as funções disponíveis são: *Count*, que resultaria em um *Card* com o número 10. *Min*, que resultaria em um *Card*

com o valor de 19. *Max*, que retornaria um *Card* com o número 41. *Avg*, que retornaria um *Card* com o número 29. *Group By,* onde o resultado é apresentado na Tabela 6. E *Histogram*, que o resultado é apresentado na Figura 4.

Tabela 6. *Group By* age: exemplo de recurso de BI para coluna obrigatória repetível do tipo Integer

age	quantity
38	2
21	2
29	3
32	1
19	1
41	1

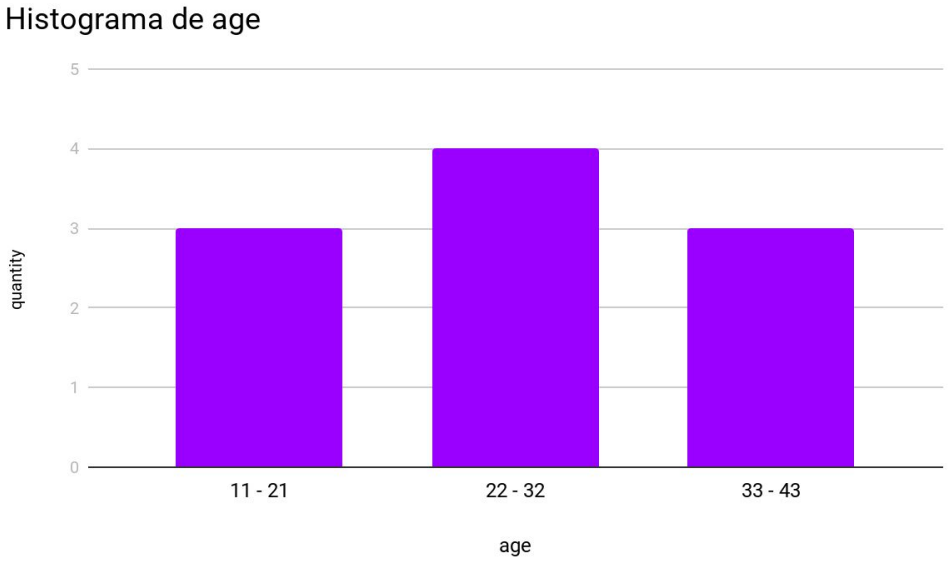

Figura 4. *Histogram* de age: exemplo de recurso de BI para coluna obrigatória repetível do tipo Integer

A coluna **height** é do tipo **Real Number** e possui a restrição OPR (Opcional Repetível). Na Figura 2, é possível reparar que para OPR as funções disponíveis são: *Count*, que resultaria em um *Card* com o número 7. *Count Null*, que resultaria em um *Card* com o número 3. E *Histogram* cujo resultado é mostrado na Figura 5.

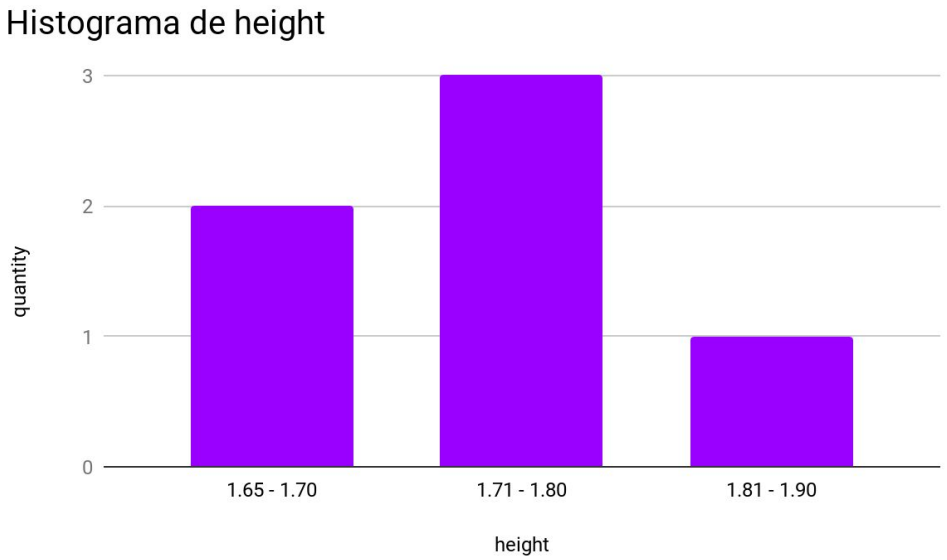

Figura 5. *Histogram* de height : exemplo de recurso de BI para coluna opcional repetível do tipo Real Number

5. Solução Proposta

Falcone e Vilar (2020) customizaram a ferramenta Metabase através da sua API REST, pela qual puderam ser invocados comandos para criar e manipular recursos de BI. Levando em consideração a afirmação dos autores foi estabelecida uma comunicação entre o sistema gerado e o Metabase, para que possa realizar a manipulação dos recursos de BI.

5.1. Implementação dos Recursos de BI no Sistema Gerado

Os recursos de BI são os resultados da combinação realizada com os metadados de uma determinada entidade. Esses recursos podem ser gráficos, tabelas e etc que surgem como resposta a questionamentos provocados em situações que exigem uma posição de um indivíduo ou equipe. Esse tipo de informação pode ser utilizada para auxiliar nas tomadas de decisões estratégicas. Um dos exemplos mais comum é quando uma empresa lança um novo produto, e para tomar a decisão de para qual público alvo irá direcionar o marketing, os responsáveis utilizam recursos de BI como referência para auxiliar na escolha.

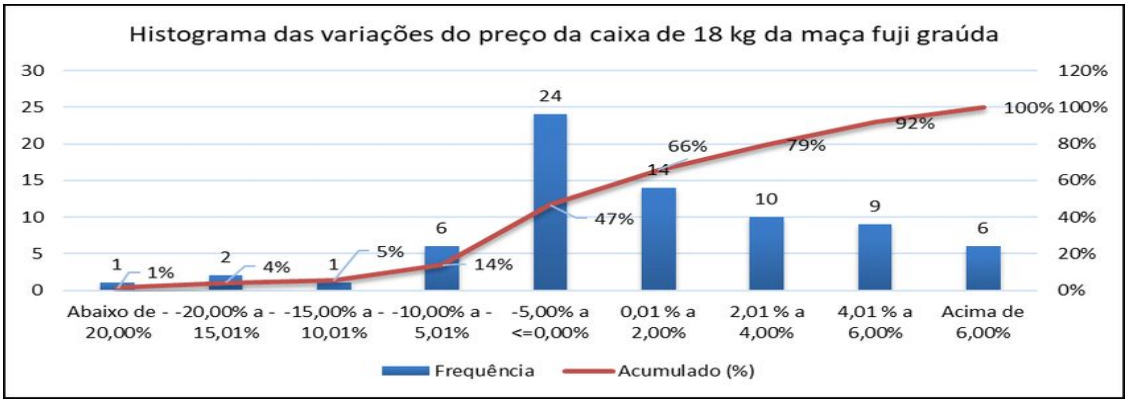

Figura 6. Exemplo de recursos de BI: *Histogram* de preço da caixa de 18 kg da maçã fuji graúda. Fonte: Site ResearchGate[6].

A Figura 6 representa um recurso de BI do tipo *Histogram*, as informações contidas nele são as variações dos preços da caixa de 18 kg da maçã fuji graúda. A partir dessas informações é possível um usuário comum, dono de um mercadinho, se basear para realizar uma projeção, com o intuito auxiliá-lo a tomar alguma decisão em relação a venda do produto.

5.1.1 Modelo do Banco de Dados para Sistema Gerado adaptado ao módulo de SSBI

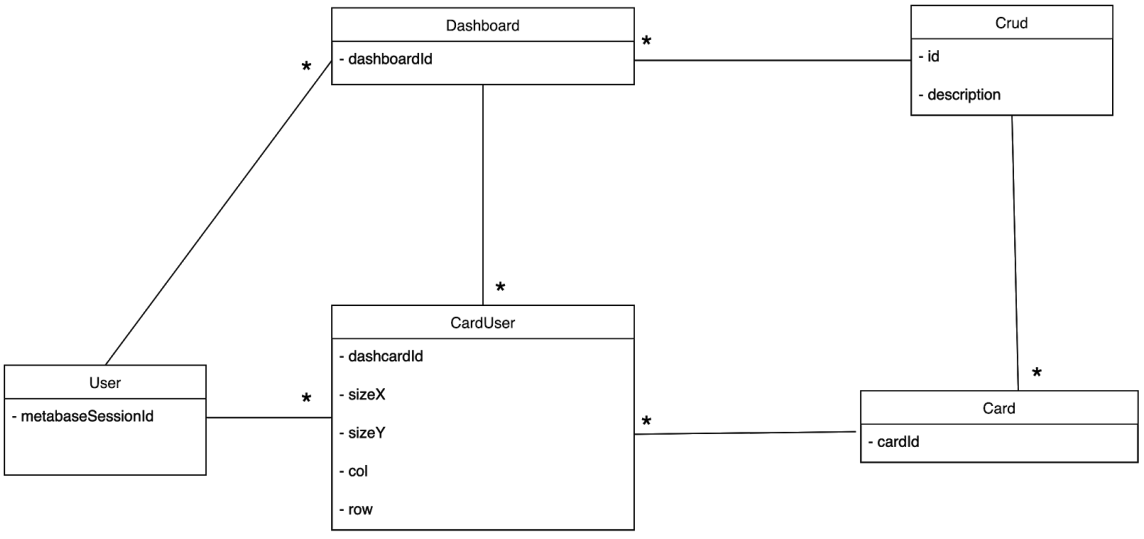

Figura 7. Diagrama UML dos Recursos de BI

Na Figura 7 é possível observar o diagrama UML dos recursos de BI no sistema gerado. Com base no diagrama percebe-se a existência de cinco entidades. São elas:

- ***User*** - consiste no usuário comum do sistema gerado;
- ***Dashboard*** - consiste no painel utilizado para agrupar os recursos de BI;

[6] Disponível em: <https://www.researchgate.net/figure/Figura-5-Histograma-dos-intervalos-de-variacao-do-preco-da-caixa-de-18-kg-da-maca-fuji_fig4_333176347>. Acesso em: 15 nov. 2020.

- ***Card*** - representa o recurso de BI;
- ***CardUser*** - consiste na representação de um *Card* dentro do *Dashboard*, no metabase é denominado como *Dashcard*;
- **Crud** - é a representação do objeto do qual estão sendo extraídos os dados para gerar os recursos de BI;

Sobre o funcionamento, os recursos de BI devem ser criados juntos ao sistema gerado, e personalizados com base nos metadados inseridos no gerador de código. Assim sendo os recursos gerados estarão inclusos na aplicação recém construída, e serão denominados como *Cards*. Deste modo o usuário pode criar *Dashboards* e inserir a eles os *Cards* desejados.

5.1.2 Incorporação dos Recursos de BI no Sistema Gerado

Para que o usuário consiga visualizar e manipular os recursos de BI no sistema gerado, foi necessário incorporar os *Cards* e *Dashboards* diretamente do Metabase, levando em consideração que a ferramenta possibilita compartilhar os recursos de forma fácil e segura.

Para realizar esse procedimento foi necessário fazer uma configuração na funcionalidade de compartilhamento dos *Cards* e *Dashboards*. A configuração consiste em gerar uma chave secreta na aba de incorporação do Metabase. E com a chave gerada é possível configurar a URL do recursos no sistema gerado, para que o *Front End* consiga incorporá-lo, visualizar sequência na Figura 8.

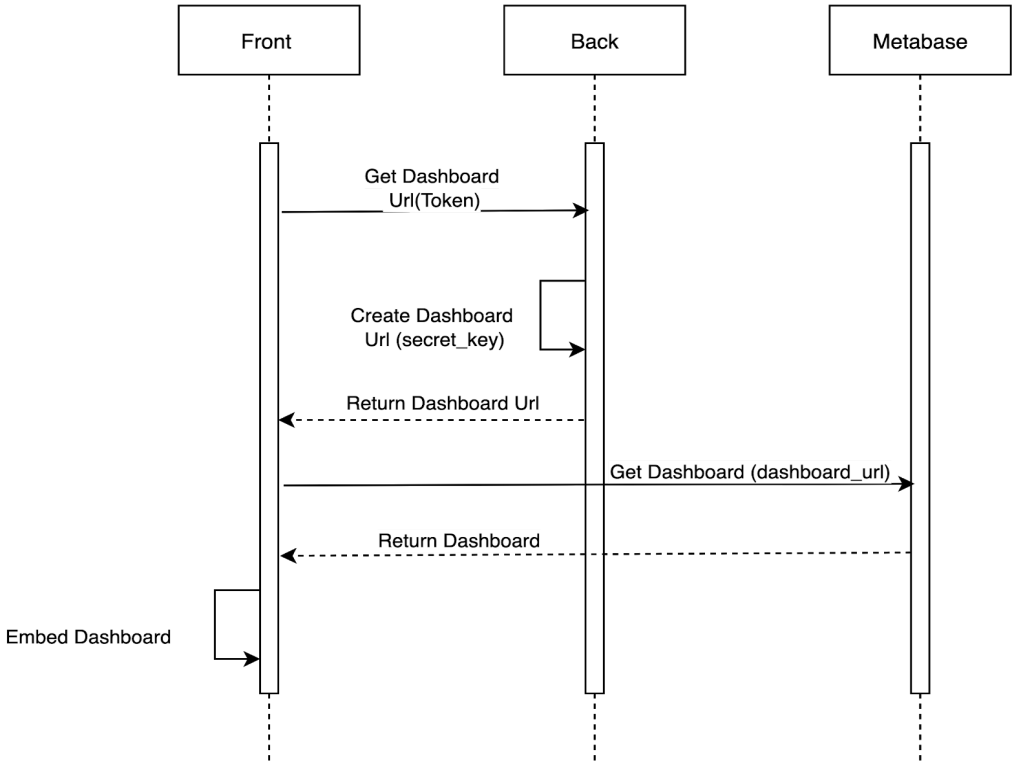

Figura 8. Diagrama de Sequência da Funcionalidade de Embed

5.2. JSONs de Configuração de *Cards*

Para que o gerador de código tenha a autonomia de produzir recursos de BI é preciso que haja comunicação com a API do Metabase, a fim de realizar requisições REST para criar e configurar os *Cards* que vão fazer parte do módulo de SSBI do sistema gerado.

Por isso foram desenvolvidos os JSONs de configuração, que serão utilizados pelo gerador de código para efetuar requisições na API do Metabase. O JSON (JavaScript Object Notation) é um formato de intercâmbio de dados leve e independente de linguagem. Portanto é comum ser utilizado para realizar esse tipo de requisição.

5.2.1 *Cards*

Para melhor entender os JSONs de configuração, foi criada a Tabela 7 que descreve a função de alguns dos seus atributos.

Tabela 7. Tabela de descrições dos atributos dos JSONs de configuração.

Atributo	Descrição
column_setting	Nesse atributo é feita a configuração da exibição do dado no *Card.*
description	Nesse atributo deve ser inserido como valor uma breve descrição sobre do que se trata o *Card.*
name	Nesse atributo deve ser inserido o título do *Card.*
display	Nesse atributo deve ser configurado o modo de exibição do *Card*.
type	Nesse atributo deve ser inserido o tipo do banco de dado, onde nesse trabalho só foi utilizado o tipo nativo.

Para melhor exemplificar os resultados obtidos nesta seção, foi inserida a Tabela 8, que representa a planilha do banco de dados no qual foram baseados os JSONs de configuração presentes no Apêndice A:

Tabela 8. Exemplo da tabela *Product*.

name	*price*	*category*
Xiaomi Redmi Note 8	1499	Smartphone
Moto G8 Power	1499	Smartphone
LG K12	699	Smartphone
Motorola One	1099	Smartphone
Galaxy M10	899	Smartphone
Galaxy A30S	1134	Smartphone
Moto G8 Play	949	Smartphone
Notebook Acer Aspire 3 A315-42G-R6FZ AMD Ryzen 5 - 8GB 1TB 15,6 Placa de Vídeo 2GB Windows 10	3134	Notebook
Notebook Samsung Essentials E30 Intel Core i3 4GB - 1TB 15,6 Full HD Windows 10	2469	Notebook
Notebook Acer Aspire 3 A315-53-3470 Intel Core i3 - 4GB 1TB 15,6 Linux	2184	Notebook

No Apêndice A, pode-se ver os detalhes dos códigos-fonte listados a seguir e que representam exemplos de Cards que podem ser gerados para os dados da Tabela 8:

1. *Card* com o valor total de registros na coluna *name*;
2. *Card* com o valor total de células nulas da coluna *name*;
3. *Card* com o preço mínimo do produto;
4. *Card* co o preço máximo do produto;
5. *Card* com a média de preços dos produtos;
6. *Card* com uma tabela agrupando a quantidade de produtos por categoria;
7. *Card* com um gráfico de *histogram*, dos intervalos de preço dos produtos.

Esses JSONs de configuração de *Cards* podem ser utilizados como referência para criar *templates* nos geradores de código a fim de automatizar a criação dos recursos de BI, implementando assim SSBI.

5.3. Autenticação

Para que seja garantida a privacidade das informações geradas no Metabase, é exigida pela aplicação que seja realizada uma autenticação na sua API. Portanto, como o sistema gerado também exige que seja realizado uma autenticação nele, foi necessário desenvolver um novo fluxo para garantir a segurança das informações geradas em ambas as aplicações.

O fluxo criado consiste em realizar duas autenticações. A primeira ocorre ainda no sistema gerado, e após ser finalizada, é efetuada uma outra autenticação na API do Metabase. Para melhor compreensão segue a Figura 9.

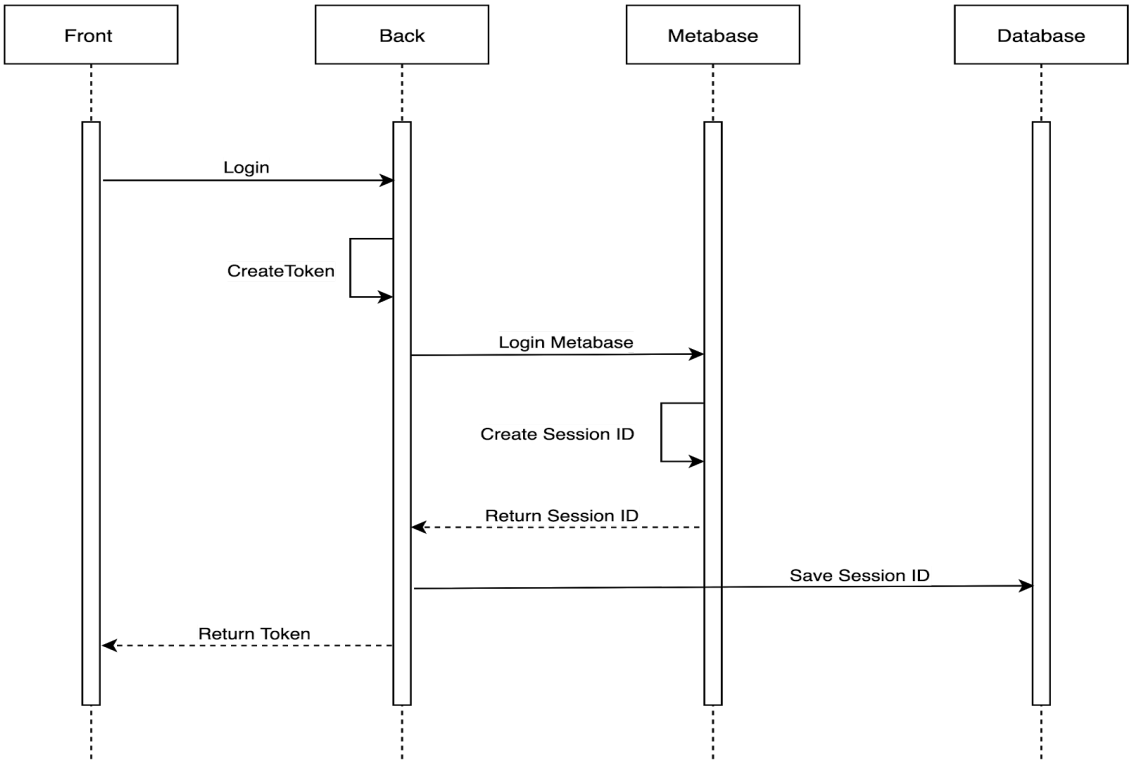

Figura 9. Diagrama de Sequência da autenticação

5.4. Arquitetura

Para chegar a um modelo final de arquitetura é essencial entender como funciona a inserção de um mecanismo para criar recursos de BI em um gerador de código baseado em Scaffolding. Na prática são adicionados ao gerador de código metadados com informações voltadas para BI, e ampliado o código fonte do template para confrontá-lo com os metadados inseridos e selecionar quais recursos de BI serão criados. E para realizar a conexão com o metabase, é estabelecida uma comunicação através da arquitetura REST, no qual é utilizado o JSON de configuração para trafegar as informações de como construir os recursos de BI até o metabase. O modelo explicado está desenhado na Figura 10.

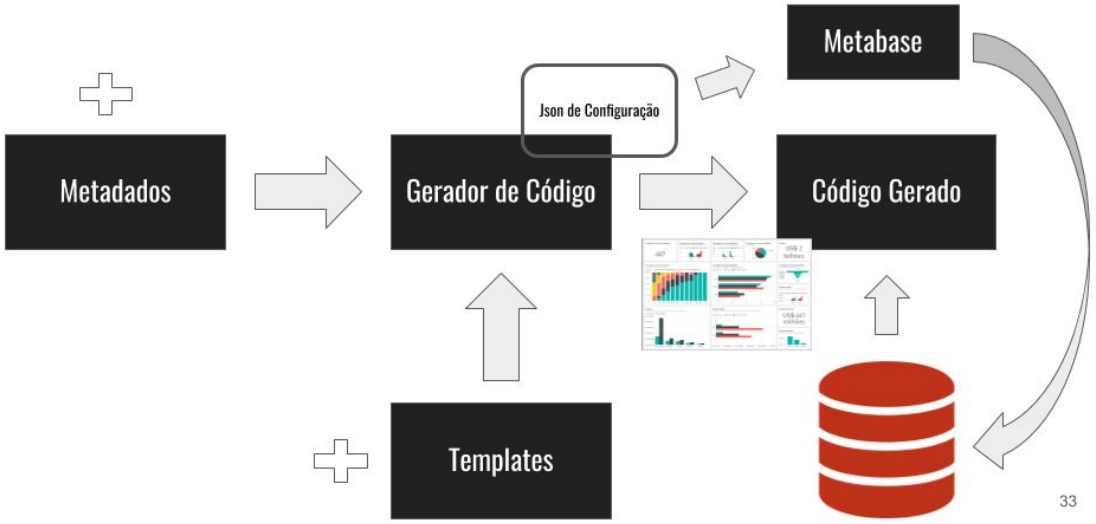

Figura 10. Arquitetura: Exemplo da adição de metadados e templates para contemplar os recursos de BI.

Por fim o diagrama de componentes na figura 11 demonstra a arquitetura alcançada a partir dos estudos realizados sobre as funções do gerador de código, metabase e sistema gerado. A fim de estabelecer um mecanismo, no qual tem como objetivo criar um sistema com um módulo de SSBI incluso.

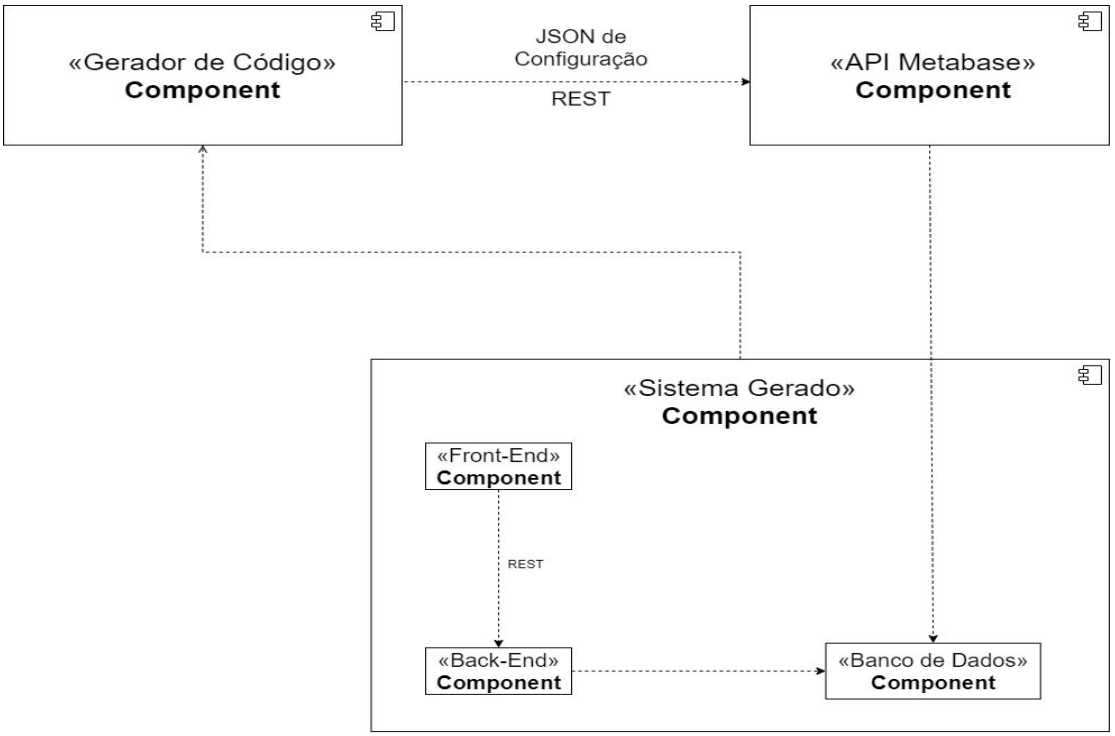

Figura 11. Arquitetura: Exemplo da comunicação entre os componentes.

6. Análise da solução em relação aos desafios de SSBI

Embora SSBI prometa mais benefícios em comparação com um sistema de BI tradicional, muitas organizações não conseguem implementá-lo na prática. Portanto Lennerholt et al. (2018) elencam em seu trabalho, "Implementation Challenges of Self Service Business Intelligence: A Literature Review", dez desafios da implementação do SSBI, relacionando-os às categorias de "Acesso e Uso de Dados" e "Usuários Autossuficientes". A seguir serão apresentadas as Tabelas 9 e 10, que tem o propósito de ilustrar os desafios abordados no trabalho citado acima. A ilustração consiste em expor os desafios e como a solução proposta lida com cada um deles, além do mais os desafios que estão marcados de verde foram cumpridos e o que está de amarelo não foi tratado no escopo do trabalho.

Tabela 9. Desafios elencados no trabalho de Lennerholt et al. (2018): Categoria de Acesso e Uso de Dados.

Desafio	Tratamento na Solução proposta
Tornar as fontes de dados fáceis de acessar e usar	As fontes de dados já estarão configuradas no sistema gerado.
Identificar os critérios de seleção de dados	A base de dados já será gerada para garantir a integridade dos dados.
Usar consultas de dados corretas	O Banco de dados estará configurado corretamente no código gerado.
Controle de integridade de dados, segurança e distribuição	Todo o controle e permissões já estarão definidos no sistema gerado.
Definir políticas para gerenciamento de dados e governança	Responsabilidade do usuário administrador, que também pode configurar os *Cards* dos demais usuários.
Preparar dados para análise visual	Os Recursos incorporados do Metabase têm toda sua estrutura visual pronta.

7. Conclusão e trabalhos futuros

A partir da combinação das restrições analisadas para os metadados, foi possível propor recursos de BI que possam ser gerados automaticamente, implementando assim o conceito de SSBI. Além disso, através das provas de conceito, descobriu-se que é possível integrar um módulo de SSBI em um sistema, criado a partir de um gerador de

código baseado em *Scaffolding* que seja capaz de se comunicar com a API do Metabase durante o processo de geração de código. Para mais, durante o processo de validação foi constatado que a solução proposta deste trabalho atende nove dos dez desafios propostos na literatura para implementar SSBI.

Como trabalhos futuros, é pretendido desenvolver UX (*User Experience*) e UI (*User Interface*) no módulo de SSBI do sistema gerado para melhor atender as necessidades do usuário, implementar a proposta de geração de código para criação de recursos de BI personalizados de acordo com os metadados em um gerador de código e habilitar a geração de recursos de BI a partir de dados externos.

Tabela 10. Desafios elencados no trabalho de Lennerholt et al. (2018): Categoria de Usuários Autossuficientes

Desafio	Tratamento na Solução proposta
Tornar as ferramentas de BI fáceis de usar	O gerador de código irá sugerir perguntas e respostas, baseadas nos metadados do domínio.
Tornar os resultados de BI fáceis de consumir e realçar	A solução é fornecer aos usuário recursos visuais prontos para facilitar a manipulação dos dados.
Fornecer as ferramentas certas para o usuário certo	A solução pode ser adicionar recursos de BI configuráveis de acordo com o papel do usuário.
Educar os usuários sobre como selecionar, interpretar e analisar dados para tomar decisões	Não foi tratado no escopo deste trabalho.

Referências

Adamus, R.; Kowalski, T. M.; Kuliberda, K.; Wi´slicki, J.; Bleja, M. Tools Supporting generation of "data-intensive"applications for a web environment. Automatyka/Akademia Górniczo-Hutnicza im. Stanislawa Staszica w Krakowie, v. 14, p. 951–960, 2010.

Alpar, Paul; Schulz, Michael. Self-service business intelligence. Business & Information Systems Engineering, v. 58, n. 2, p. 151-155, 2016.

Braghittoni, Ronaldo. Business Intelligence: Implementar do Jeito Certo e a custo zero. Editora Casa do Código, 2017.

Coronel, C.; Morris, S.; Rob, P. Database Systems. Cengage learning, USA, 2010.

Costa, A. Desenvolvimento de componentes de software para aplicações web utilizando o framework ruby on rails, 2018.

Date, C. J. Introdução a Sistemas de Banco de Dados. Ed. Campus, 2000. 7a Edição.

Elmasri, R et al. Sistemas de banco de dados. São Paulo: Pearson Addison Wesley, 2005.

Falcone, José; Vilar, Rodrigo. Avaliação de ferramentas para implantação automática de Business Intelligence. 2020.

Franky, M. Pavlich-Mariscal, J.Improving implementation of code generators: A regular-expression approach. In: Informatica(CLEI), 2012XXXVIIIConferencia Lation americana En. [S.1.: s.n.], 2012. p. 1–10.

Herrington, J. Code generation in action. Manning Publications Co., 2003.

Ikematu, Ricardo Shoiti. Gestão de metadados: sua evolução na tecnologia da informação. DataGramaZero-Revista de Ciência da Informação, v. 2, n. 6, 2001.

Lazzaretti, A. T. et al. XDC: uma proposta de controle de restrições de integridade de domínio em documentos XML. 2005.

Lennerholt, C., Van Laere, J., & Söderström, E. (2018, January). Implementation challenges of self service business intelligence: A literature review. In Proceedings of the 51st Hawaii International Conference on System Sciences.

Negash, S., Gray, P. Business Intelligence, 2008.

Niso. Understanding metadata. Bethesda, MD, 2004. ISBN 1-880124-62-9.

Pressman, R. S. Engenharia de Software. 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

Sasidharan, D. K. Full Stack Development with JHipster: Build modern web applications and microservices with Spring and Angular. Packt Publishing Ltd, 2018.

Silberchatz, A; Korth, H. F; Sudarshan, S. Sistemas de Banco de Dados. Ed. Makron Books, 1999. 3a Edição.

Verma, A. Mvc architecture: A comparative study between ruby on rails and laravel. Indian Journal of Computer Science and Engineering (IJCSE), 5(5), 196-198, 2014.

Apêndice A

Código Fonte 1. JSON de configuração: Cria um *Card* que retorna o resultado da função Count sobre a coluna *name* da Tabela 8.

```
{
  "visualization_settings": {},
       "description": "Count",
       "collection position": null,
       "result metadata": null,
       "metadata checksum": null,
       "collection id": null,
       "name": "Count",
       "display": "scalar",
       "dataset query": {
              "database": 5,
              "type": "native",
              "native": {
                     "query" : "SELECT count(name) AS count FROM
public.product"
              }
       }
}
```

Código Fonte 2. JSON de configuração: Cria um *Card* retorna o resultado da função Count Null sobre a coluna *name* da Tabela 8.

```
{
  "visualization_settings": {},
       "description": "Count Null",
       "collection_position": null,
       "result_metadata": null,
       "metadata_checksum": null,
       "collection_id": null,
       "name": "Count Null",
       "display": "scalar",
       "dataset_query": {
              "database": 5,
              "type": "native",
              "native": {
                     "query" : "SELECT count(*) - count(name) as count FROM
product"
              }
       }
}
```

Código Fonte 3. JSON de configuração: Cria um *Card* que retorna o resultado da função Min sobre a coluna *price* da Tabela 8.

```
{
  "visualization_settings": {
             "column_settings": {
      "[\"name\",\"min\"]": {
        "number_style": "decimal",
        "number_separators": ",."
      }
    }
       },
       "description": "Preço Mínimo",
       "collection position": null,
       "result metadata": null,
       "metadata checksum": null,
       "collection id": null,
       "name": "Min",
       "display": "scalar",
       "dataset query": {
             "database": 5,
             "type": "native",
             "native": {
                   "query" : "SELECT min(public.product.price) AS min FROM
public.product"
             }
       }
}
```

Código Fonte 4. JSON de configuração: Cria um *Card* que retorna o resultado da função Max sobre a coluna *price* da Tabela 8.

```
{
  "visualization_settings": {
             "column_settings": {
      "[\"name\",\"max\"]": {
        "number_style": "decimal",
        "number_separators": ",."
      }
    }
       },
       "description": "Preço Máximo",
       "collection_position": null,
       "result_metadata": null,
       "metadata_checksum": null,
       "collection_id": null,
       "name": "Max",
       "display": "scalar",
       "dataset_query": {
             "database": 4,
             "type": "native",
             "native": {
                   "query" : "SELECT max(public.product.price) AS max FROM
public.product"
             }
       }
}
```

Código Fonte 5. JSON de configuração: Cria um *Card* que retorna resultado da função Avg sobre a coluna *price* da Tabela 8.

```
{
  "visualization_settings": {
              "column_settings": {
      "[\"name\",\"avg\"]": {
        "number_style": "decimal",
        "number_separators": ",."
      }
    }
       },
       "description": "Média de Preços",
       "collection position": null,
       "result metadata": null,
       "metadata checksum": null,
       "collection id": null,
       "name": "Avg",
       "display": "scalar",
       "dataset query": {
              "database": 4,
              "type": "native",
              "native": {
                     "query" : "SELECT avg(public.product.price) AS avg FROM
public.product"
              }
       }
}
```

Código Fonte 6. JSON de configuração: Cria um *Card* com o Group By da coluna *category,* por produtos da Tabela 8.

```
{
  "visualization_settings": {},
       "description": "Count de Produtos por Categoria",
       "collection_position": null,
       "result_metadata": null,
       "metadata_checksum": null,
       "collection_id": null,
       "name": "Count de Produtos por Categoria",
       "display": "table",
       "dataset_query": {
              "database": 5,
              "type": "native",
              "native": {
                     "query" : "SELECT category as \"Categoria\",
count(category) as \"Quantidade de Produtos\" FROM product GROUP BY
category"
              }
       }
}
```

Código Fonte 7. JSON de configuração: Cria um *Card* com o *Histogram* da coluna *price.*

```
{
  "description": "Histograma da quantidade de produtos em função do
intervalo de preços.",
  "collection_position": null,
  "database id": 5,
  "enable embedding": true,
  "collection id": null,
  "query type": "native",
  "name": "Histograma de Preço",
  "dataset query": {
    "database": 5,
    "type": "native",
    "native": {
      "query": "SELECT ((floor(((public.product.price - 500.0) / 125.0)) *
125.0) + 500.0) AS price, count(*) AS count FROM public.product GROUP BY
((floor(((public.product.price - 500.0) / 125.0)) * 125.0) + 500.0) ORDER
BY ((floor(((public.product.price - 500.0) / 125.0)) * 125.0) + 500.0) ASC"
    }
  },
  "display": "bar",
  "visualization settings": {
    "graph.show goal": false,
    "graph.y axis.title text": "Quantidade de Produtos",
    "graph.show values": true,
    "graph.y_axis.max": 10,
    "graph.x_axis.axis_enabled": true,
    "graph.x_axis.title_text": "Intervalo de Preços",
    "graph.label_value_frequency": "fit",
    "graph.metrics": [
      "count"
    ],
    "column_settings": {
      "[\"name\",\"price\"]": {
        "number_separators": ",."
      }
    },
    "series_settings": {
      "count": {
        "display": "bar",
        "color": "#A989C5"
      }
    },
    "graph.y_axis.auto_range": false,
    "graph.x_axis.scale": "histogram",
    "graph.dimensions": [
      "price"
    ],
    "stackable.stack_type": null
  },
  "collection": null
}
```