

**UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

EDINEI JACOB BAUER

**GERAÇÃO DE INTERFACES DE USUÁRIO PARA OPERAÇÕES CRUD COM
BASE EM METADADOS**

CRICIÚMA

2018

EDINEI JACOB BAUER

**GERAÇÃO DE INTERFACES DE USUÁRIO PARA OPERAÇÕES CRUD COM
BASE EM METADADOS**

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Graduação no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Fabrício Giordani

CRICIÚMA

2018

EDINEI JACOB BAUER

**GERAÇÃO DE INTERFACES DE USUÁRIO PARA OPERAÇÕES CRUD
COM BASE EM METADADOS**

Trabalho de Conclusão de Curso
aprovado pela Banca Examinadora para
obtenção do Grau de bacharel, no Curso
de Ciência da Computação da
Universidade do Extremo Sul
Catarinense, UNESC, com Linha de
Pesquisa em Desenvolvimento para Web

Criciúma, 28 de Junho de 2018

BANCA EXAMINADORA


Prof. Fabrício Giordani - Esp. - (Universidade UNESC) - Orientador


Prof. Anderson Rodrigo Farias - Esp. - (Faculdade SATC/UNIBAVE)


Prof. Gustavo Bisognin - Me. - (Universidade UNESC)

A minha família e as pessoas que próximas.

AGRADECIMENTOS

Agradeço a todas as pessoas que estiveram junto comigo nesta fase da minha vida e me apoiaram a todo o momento para que eu tivesse êxito na conclusão do curso de Ciência da Computação.

Agradeço principalmente aos meus pais pela dedicação e apoio ao longo da minha vida e sempre me incentivaram a realizar meus sonhos. E também a minha namorada que sempre esteve por perto para dar apoio durante o desenvolvimento do trabalho, realizando testes na ferramenta e colaborando com compras de matérias. Também agradeço ao meu orientador Fabrício Giordani pelo apoio, conhecimento e pela ajuda com ferramentas contra plágio.

“O prazer no trabalho aperfeiçoa a obra.”

Aristóteles

RESUMO

No desenvolvimento de sistemas web, é comum o armazenamento de dados persistente utilizando banco de dados. Para o gerenciamento dos dados armazenados no sistema, Interfaces de Usuários (UI, User Interface) para operações de Inserção, Obtenção, Exclusão e Atualização (CRUD - *Create, Retrieve, Update and Delete*) de registros são utilizadas, representando até 30% do desenvolvimento de um sistema. Objetivando automatizar completamente o desenvolvimento de UI para operações CRUD, este trabalho propõe a criação de uma ferramenta para geração de códigos baseada em modelos. Para alcançar este objetivo, a ferramenta utiliza um motor de templates para receber variáveis através dos dicionários de metadados e aplica a modelos de templates predefinidos no sistema. Completando sua arquitetura, arquivos de controle e estilo são acoplados a interface de usuário retornada pela ferramenta inibindo o desenvolvimento de código-fonte e da refatoração ao longo da evolução do sistema. A ferramenta se limita a tipos de dados presentes no sistema para a geração de UI para operações CRUD, sendo necessário a implementação de novos tipos de dados, templates, controles, estilos e mapeamento objeto-relacional (MOR) para atender a derivadas necessidades presentes no mercado. Focando inicialmente na linguagem de programação *Hypertext Preprocessor* (PHP), o trabalho proposto aponta uma ferramenta escalável e apta a ser utilizada em sistemas profissionais de desenvolvimento para a geração automatizada de UI para operações CRUD.

Palavras-chave: *operações CRUD, geração de códigos, interface de usuário, metadados, dicionário de metadados.*

ABSTRACT

In the development of web systems, it's common to store persistente data in database. For managing the data stored in the system, User Interfaces (UI) are used to create, retrieve, update and delete (CRUD) records, representing up to 30% of the development of a system. In order to fully automate the development of the UI for CRUD operations, this work offers a tool for model-based data generation. To achieve this goal, the tool uses a template engine to receive variables through metadata dictionaries and applies to predefined templates on the system. Completing its architecture, control files and style are coupled to the user interface returned by the tool, inhibiting the development of source code and refactoring throughout the evolution of the system. The tool is limited to data types present in the system for generating UI for CRUD operations, requiring the implementation of new data types, templates, controls, styles and Object-Relational Mapping (ORM) to meet the needs derived from Marketplace. Focusing initially on the Hypertext Preprocessor (PHP) programming language, the proposed work points to a scalable tool that can be used in professional development systems for the automated generation of UI for CRUD operations.

Keywords: CRUD operations, code generation, user interface, metadados, metadata dictionary.

LISTA DE ILUSTRAÇÕES

Figura 1 – Representação Server-side e Client-side.....	21
Figura 2 – Exemplos de tags HTML.	23
Figura 3 – Comportamento da ferramenta ao ser executada	27
Figura 4 – Estrutura dos diretórios que armazenam os dicionários de metadados ...	28
Figura 5 – Modelo de Dicionário de metadados	29
Figura 6 – Modelo de um arquivo de informação sobre o dicionário de metadados .	30
Figura 7 – Tipo de dado <i>list</i> representado na UI para operações CRUD	33
Figura 8 – Entidades necessárias para o exemplo da empresa de jornalismo.....	37
Figura 9 – Chamada da ferramenta em um arquivo PHP	39
Figura 10 – Tabela noticias gerada no SGBD MySQL pela ferramenta	39
Figura 11 – Tabela categorias gerada no SGBD MySQL pela ferramenta.....	39
Figura 12 - Tabela de relação entre noticias e categorias gerada no SGBD pela ferramenta.....	40
Figura 13 – Diretório do Dicionário de Metadados	40
Figura 14 – UI gerada pela ferramenta para operações CRUD na entidade <i>noticias</i>	41
Figura 15 – Erro retornado e apresentado na UI ao submeter o formulário	42
Figura 16 – Cadastro de registro realizado com sucesso na entidade <i>noticias</i>	43
Figura 17 – Registro inserido no SGBD na tabela <i>noticias</i> após cadastro submetido	43
Figura 18 – Chamando UI para operações CRUD em modo de edição.....	44
Figura 19 – UI para operações CRUD em modo de edição com botão para exclusão do registro	44

LISTA DE ABREVIATURAS E SIGLAS

AJAX	<i>Asynchronous Javascript and XML</i>
BD	Banco de Dados
BDR	Banco de Dados Relacional
CRUD	<i>Create, Retrieve, Update and Delete</i>
CSS	<i>Cascade Style Sheet</i>
CSS3	<i>Cascade Style Sheet</i>
HTML	<i>HyperText Markup Language</i>
HTML5	<i>HyperText Markup Language</i>
JS	<i>JavaScript</i>
JSON	<i>JavaScript Object Notation</i>
MOR	Mapeamento Objeto-Relacional
ORM	Object-Relational Mapping
NoSQL	<i>Not Only SQL</i>
OO	Orientado a Objetos
PDF	<i>Portable Document Format</i>
PHP	<i>Hypertext Preprocessor</i>
POO	Programação Orientada a Objetos
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SQL	<i>Structured Query Language</i>
UI	<i>User Interface</i>
W3C	<i>World Wide Web Consortium</i>
XML	<i>Extensible Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO	12
1.1 OBJETIVO GERAL	13
1.2 OBJETIVOS ESPECÍFICOS	13
1.3 JUSTIFICATIVA	13
1.4 ESTRUTURA DO TRABALHO	14
2 REVISÃO DE LITERATURA	16
2.1 DADOS	16
2.2 METADADOS	16
2.3 MOTORES DE TEMPLATES	17
2.4 PERSISTÊNCIA DE DADOS	18
2.4.1 Banco de dados e SGBD	18
2.4.1.1 Banco de dados não relacional	18
2.4.1.2 Banco de dados relacional	19
2.4.2 Operações CRUD e mapeamento objeto-relacional	20
2.5 JSON	20
2.7 DESENVOLVIMENTO PARA A WEB	21
2.7.1 Front-end	22
2.7.1.1 HTML5	22
2.7.1.2 CSS3	23
2.7.1.3 JavaScript	24
2.7.2 Back-end	24
2.8 TRABALHOS CORRELATOS	24
3 GERAÇÃO DE UI PARA OPERAÇÕES CRUD	26
3.1 VISÃO GERAL	26
3.2 FUNCIONAMENTO DA FERRAMENTA	26
4 APRESENTAÇÃO E ANÁLISE DOS DADOS	36
4.1 CONFIGURAÇÕES INICIAIS	36
4.1 CENÁRIO	36
4.2 TIPOS DE DADOS UTILIZADOS	37
4.3 DESENVOLVIMENTO DOS DICIONÁRIOS DE METADADOS	38
4.4 GERANDO A UI PARA OPERAÇÕES CRUD	38
4.5 UTILIZANDO A UI PARA OPERAÇÕES CRUD	41

4.6 EDITANDO E EXLUINDO REGISTROS	43
4.7 RESULTADOS OBTIDOS	45
5 CONCLUSÃO	46
REFERÊNCIAS.....	48
APÊNDICE A – ARTIGO.....	49
APÊNDICE B – TIPOS DE DADOS.....	61
APÊNDICE C – TEMPLATES.....	64
APÊNDICE D – DICIONÁRIO DE METADADOS.....	77

1 INTRODUÇÃO

No desenvolvimento de sistemas web, é comum a necessidade de armazenamento de dados. Não somente o armazenamento quanto o controle destes dados é importante para a gestão da informação. Nestes casos, interfaces de usuários são utilizadas para a realização de operações CRUD nos registros armazenados.

O desenvolvimento de UI para operações CRUD pode representar até 30% do tempo gasto necessário para o desenvolvimento de um sistema web (MRACK, 2007), sendo que, neste tipo de sistema, existe uma preocupação maior em relação a disponibilidade e prazos para desenvolvimento. No desenvolvimento de UI para operações CRUD, o processo é repetitivo e constantemente modificado, portanto, propenso a erros.

Este trabalho propõe o desenvolvimento de uma ferramenta web para a geração automatizada de UI para operações CRUD com base em metadados, auxiliando através da geração de códigos para maior produtividade no desenvolvimento de sistemas web.

Para o desenvolvimento desta ferramenta, os dados armazenados serão associados a entidades, no qual possuirá um dicionário de metadados descrevendo sua estrutura, relação e composição dos dados. Através desse dicionário de metadados, a ferramenta validará os dados submetidos pela UI, resultando em consistência nos dados armazenados ou em respostas personalizadas sobre os erros encontrados nos dados. Ainda com o dicionário, será possível gerar os campos de entrada de dados conforme a necessidade de armazenamento da informação.

O processo de automação é uma tendência natural (AREND, 2011), que garante padronização e produtividade. A geração automática de UI para operações CRUD em tempo de execução torna o controle dos dados flexível, permitindo alterações em sua estrutura, relação e definição sem comprometer o funcionamento do sistema, garantindo um custo menor para a produção e manutenção do sistema web.

Nas últimas 3 décadas, diversos projetos foram desenvolvidos, entretanto, de forma geral, ainda não foi obtido os resultados desejados, os quais se referem principalmente à facilidade de uso, rapidez de desenvolvimento e integração ao ambiente de programação (PUERTA e EISENSTEIN, 1999, tradução nossa). Arend

(2011) comprova através de seu trabalho de conclusão de curso que este tipo de ferramenta traz benefícios para o desenvolvimento, assim como a dificuldade que é implementar e manter o seu uso.

1.1 OBJETIVO GERAL

Desenvolver uma ferramenta para gerar de forma automática interface de usuário para operações CRUD com base em metadados.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do trabalho são:

- a) Definir modelo do dicionário de metadados;
- b) Autocompletar metadados ausentes com valores padrões para cada tipo de atributo existente no dicionário de metadados;
- c) Definir motor de templates a ser utilizado;
- d) Criar templates para cada tipo de atributo diferente no dicionário;
- e) Definir controladores e estilos a serem utilizados nas UI;
- f) Validar dados submetidos ao banco de dados com base nos metadados;

1.3 JUSTIFICATIVA

No desenvolvimento de sistemas web, geralmente existe uma preocupação maior em cumprir os prazos dentro do tempo previsto, a utilização de metas é comumente utilizada para controle do progresso do desenvolvimento do sistema.

Em sistemas que acessam dados, é comum a necessidade de uma UI para realizar operações CRUD em cada entidade do sistema. Segundo Mrack (2007), o desenvolvimento de UI para operações CRUD representam até 30% do tempo gasto no desenvolvimento de sistemas. O processo de desenvolvimento de UI para operações CRUD acaba tornando-se repetitivo, e por consequência, propenso a erros. Eventualmente, com a evolução do sistema, as UI para operações CRUD necessitam de atualizações.

A fim de melhorar a produtividade das equipes, uma tendência natural é a busca por automação de alguns processos realizados (AREND, 2011). O objetivo deste trabalho é possibilitar, com a criação de uma ferramenta, a geração de UI para operações CRUD. Para isso, pretende-se utilizar um motor de templates alimentado por dicionários de metadados para gerar modelos de código-fonte. Sabendo que os metadados são os dados que descrevem a estrutura dos dados (TAYLOR, 2016), é possível saber não só a informação de um campo da entidade, mas também o tipo e as especificações do campo. Desta forma, a ferramenta possibilita o auxílio no desenvolvimento de um sistema web na criação de UI para operações CRUD.

Com o uso do dicionário de metadados como base para a construção da estrutura de modelo de dados de uma entidade no SGBD, pode-se obter todos os requisitos necessários para validar as informações submetidas para o armazenamento. Inibindo a necessidade de validação dos dados.

1.4 ESTRUTURA DO TRABALHO

Este trabalho é dividido em cinco capítulos. Sendo o primeiro capítulo, a apresentação da introdução, objetivos gerais e específicos, assim como a justificativa.

O texto remanescente está organizado em 5 capítulos. No capítulo 2, é apresentado uma revisão bibliográfica com os conceitos utilizados, abordando tópicos como dicionário de metadados, o uso de motores de templates para geração de códigos, operações CRUD, Mapeamento Objeto-Relacional, persistência de dados, desenvolvimento para a web e trabalhos correlatos.

O capítulo 3 apresenta o trabalho proposto, indicando os métodos utilizados para o desenvolvimento. Neste capítulo será demonstrado a criação dos dicionários de metadados, os modelos de templates utilizados, o controle dos dados, e a persistência dos dados das entidades.

No capítulo 4 é apresentado um experimento da ferramenta, onde é realizado um teste de uso da ferramenta sendo aplicado em uma necessidade real para uma empresa de pequeno porte na área de jornalismo, por fim se obtém os resultados do experimento.

Por último, no capítulo 5 apresenta as conclusões obtidas com o desenvolvimento da ferramenta para geração automatizada de UI para operações

CRUD. Após é indicado sugestões de trabalhos futuros para melhorias da ferramenta desenvolvida.

2 REVISÃO DE LITERATURA

Este capítulo apresenta a definição de alguns conceitos utilizados no desenvolvimento Web para uma melhor compreensão dos fundamentos estudados e utilizados na aplicação proposta, bem como uma análise de ferramentas de auxílio na realização da geração de códigos.

2.1 DADOS

Em informática, dados são representados por um conjunto de caracteres, podendo ser letras, números, imagens ou símbolos que representam um evento no tempo e espaço (SOMASUNDARAM; SHRIVASTAVA; SERVICES, 2009). Existem dois tipos de dados, estruturados e não estruturados. Dados não estruturados não possuem uma formatação específica e são difíceis de serem processados, são exemplos: imagens, mensagens de e-mail, documentos de texto. Já os dados estruturados são devidamente formatados e organizados em tabelas, linhas e colunas, de fácil processamento, e normalmente é utilizado um Sistema Gerenciador de Banco de Dados (SGBD) (SHRIVASTAVA; SOMASUNDARAM, 2009).

2.2 METADADOS

Metadados é qualquer dado que tem como objetivo descrever outro dado (AREND, 2011; SOUZA; CATARINO; SANTOS, 1997). Metadados trazem contexto aos dados, permitindo que sejam manipulados dentro de uma aplicação. A quantificação de metadados define o quanto o dado será preciso e amplo, permitindo um maior controle de uso. Dados bem descritos pelo uso de metadados, podem ser utilizados em múltiplos interesses e torna a recuperação e compartilhamento de recursos heterogêneos mais eficientes (ALVES, 2010; SOUZA; CATARINO; SANTOS, 1997).

A principal finalidade dos metadados é documentar e organizar de forma estruturada e relacional os dados, permitindo assim, colaborar com o desenvolvimento ágil, minimizar esforços, duplicações e facilitar a manutenção dos dados (SOUZA; CATARINO; SANTOS, 1997). Para a organização dos metadados, normalmente faz-se uso de um dicionário de metadados.

Dicionário de metadados é um conjunto de metadados que definem uma entidade. Não existe um padrão de dicionário de metadados, as informações são de livre escolha (SOUZA; CATARINO; SANTOS, 1997). O dicionário de metadados conterá uma seção principal com os metadados assim como as descrições de cada dado, poderá conter uma seção descrevendo como os dados são subdivididos, relacionados e quais convenções são adotadas em sua definição. São exemplos de metadados que poderiam ser atribuídos a um dado: nome do campo, tipo de dado, tamanho, formato, obrigatoriedade, valor padrão, etc. (RIBEIRO, 1996).

Se um sistema de arquivos simples não possuir um dicionário de metadados, os aplicativos escritos para operar sobre estes arquivos devem conter o equivalente aos metadados como parte da aplicação (TAYLOR, 2016). Os dicionários de metadados podem ser utilizados em conjunto com motores de templates para fornecer variáveis de entrada.

2.3 MOTORES DE TEMPLATES

Dentro do contexto de aplicações web, um motor de templates é um software que substitui marcações existentes em um arquivo de template por conteúdos dinamicamente carregados pela aplicação (GARCÍA; CASTANEDO; FUENTE, 2007, tradução nossa). Um template é utilizado quando se tem uma mesma estrutura *HyperText Markup Language* (HTML), sendo aplicada mais que uma vez. Para evitar a reescrita, simplificar a aplicação e facilitar a manutenção, apenas às informações são marcadas para serem substituídas dinamicamente pelo motor de templates, mantendo o formato do contexto. Os motores de templates tem foco na substituição de conteúdo sem formatação, porém, geralmente apresentam algumas características similares às linguagens de programação de alto nível, tais como condicionamento e laços de repetições (AREND, 2011). Isso evita que o template possua repetições de blocos dentro do próprio template, e mantém as variáveis limpas de código HTML.

Os templates surgiram da necessidade de arquivos dinamicamente transformados com o intuito de facilitar o desenvolvimento, melhorar a flexibilidade e reduzir custos de manutenção. São aplicados principalmente em documentos, e-mails, PDF (*Portable Document Format*) e páginas HTML (PARR, 2004, tradução nossa). O que resultou em diversos softwares de motores de templates disponíveis

atualmente no mercado. São exemplos de *softwares* de motores de templates para a linguagem PHP, o *Smarty, Twig, Blade, Mustache, Volt, Foil* entre outros.

2.4 PERSISTÊNCIA DE DADOS

A persistência de dados é normalmente necessária nos sistemas web, seja ela feita pelo uso de banco de dados, arquivos *Extensible Markup Language* (XML) ou *JavaScript Object Notation* (JSON), de forma que a restauração dos dados para a memória do sistema possa ocorrer sem perda da informação (COELHO; SARTORELLI, 2004). Portanto, a persistência dos dados, é uma forma de assegurar que os dados utilizados por um sistema serão duráveis, através de uma forma de armazenamento estável que possibilita a restauração posterior.

2.4.1 Banco de dados e SGBD

Banco de Dados (BD) é uma coleção de dados persistentes inter-relacionados, representando informações sobre uma entidade (DATE, 2004, p. 10; CUER, 2014). Um SGBD é um conjunto de softwares para gerenciar o banco de dados, projetado para trabalhar com grandes volumes de informação. A definição da estrutura de armazenamento e dos mecanismos para manipulação das informações é fundamental para o gerenciamento dos dados através do SGBD, que também é responsável pela segurança, acesso e integridade dos dados armazenados (SILBERSCHATZ; KORTH; SUDARSHAN, 1999, tradução nossa).

2.4.1.1 Banco de dados não relacional

Segundo Lóscio et al. (2011), banco de dados não relacional surgiu da necessidade de aprimorar a performance ao lidar com grande volume de dados semiestruturados ou não estruturados, com necessidade de alta disponibilidade e escalabilidade, sendo utilizado pela primeira vez em 1998. O termo NoSQL (*Not Only SQL*) é uma notação aos bancos de dados não relacionais (SADALAGE et al., 2013).

Banco de dados não relacional diferentes dos BDR, não utilizam tabelas como forma de apresentar os dados, e possuem métodos de acesso diferente do

Structured Query Language (SQL). A principal característica do NoSQL, é a falta de esquema, o que significa que um BD de documentos não cria associação com colunas ou tipo de dados predefinido. Além disso, os atributos de um documento são livres de definição e podem receber quantos campos forem necessários (MEDEIROS, 2015; SADALAGE et al., 2013).

Em bancos de dados NoSQL orientados a documento os dados são armazenados em uma coleção (análogo a tabela), que recebe documentos (análogo a linhas) e possui atributos (análogo a colunas). Um documento representa um objeto com um identificador único, com um conjunto de campos livres, podendo ser *strings*, listas, ou documentos aninhados (LÓSCIO et al., 2013). Os documentos são estruturados na forma de árvores hierárquicas e auto descritivas, construídas de coleções, mapas e valores escalares, sendo armazenados pelo banco de dados no formato JSON, XML, entre outros (MEDEIROS, 2015). A flexibilidade de possuir documentos com atributos distintos, sem qualquer definição ou restrição, possibilita novas formas de armazenamento, sendo uma característica de muitos bancos de dados NoSQL (PROJETAND..., 2013).

Segundo Sadalage et al. (2013), quatro categorias se destacam no uso de ecossistemas NoSQL: Chave-valor, Família de Colunas, Documento e Grafos. As três primeiras categorias são chamadas de *orientação agregada*, pois trabalham com dados na forma de unidades, com estrutura mais complexa do que um conjunto de tuplas.

2.4.1.2 Banco de dados relacional

Banco de Dados Relacional (BDR) é um método de persistência de dados estável, segura e rápida (COELHO; SARTORELLI, 2004). Os Sistemas Gerenciadores de Bancos de Dados Relacional (SGBDR) possuem um sistema robusto, com suporte a grande volume de dados, transações, controle de concorrência e otimizações. São largamente utilizados no mundo todo, e dominam o mercado de sistemas, são exemplos de SGBDR: Mysql, MariaDB, PostgreSQL, SQL Server e Oracle.

O modelo relacional armazena as informações em tabelas unidimensionais, com linhas e colunas, onde cada linha da tabela representa um registro no bando de dados. Os dados sobre a identidade dos objetos, seu

relacionamento (associação, agregação e herança) e seu estado devem ser armazenados nas tabelas relacionais do SGBDR (COELHO; SARTORELLI, 2004).

No Paradigma da Orientação a Objetos (OO), a persistência dos dados, significa a persistência dos objetos. O conflito entre o modelo relacional e orientado a objetos dificulta na persistência manual dos objetos no SGBDR, tornando o processo mais complexo e resultando na inconsistência com o modelo OO (COELHO, 2004; SARTORELLI, 2004). A técnica MOR trabalha na conversão de uma representação de dados em outra, utilizando técnicas que traduzem os diferentes esquemas de dados, tornando possível a persistência de um modelo de objeto em um modelo de dados.

2.4.2 Operações CRUD e mapeamento objeto-relacional

CRUD é um acrônimo para as operações de inserção, obtenção, atualização e exclusão de dados necessárias para manipular qualquer dado persistente (AREND, 2011).

Na Programação Orientada a Objetos (POO), cada entidade de um sistema é representada por um objeto com métodos e atributos, que geralmente manipulam valores não escalares, diferente do modelo de dados relacional no formato tabular, onde as tabelas armazenam valores de forma escalar. Essa incompatibilidade pode ser amenizada com o uso de um mapeamento dos objetos (AREND, 2011). Com o uso da técnica MOR, um objeto pode realizar as operações CRUD em um BDR através de métodos, minimizando a impedância que existe entre um modelo relacional e um modelo de objeto.

2.5 JSON

JSON é um formato de texto derivado de objetos em JavaScript, para a serialização de dados estruturados entre linguagens de programação definido a partir da terceira geração do *ECMAScript Programming Language Standard* (ECMA-262) (BRAY, 2014, tradução nossa). JSON foi desenhado dentro de um modelo que fosse mínimo, portátil, textual e um subconjunto de JavaScript, para ser facilmente lido e escrito por seres humanos, e simples de ser interpretado e gerado por máquinas (ECMA INTERNATIONAL, 2013, tradução nossa). Não depende de

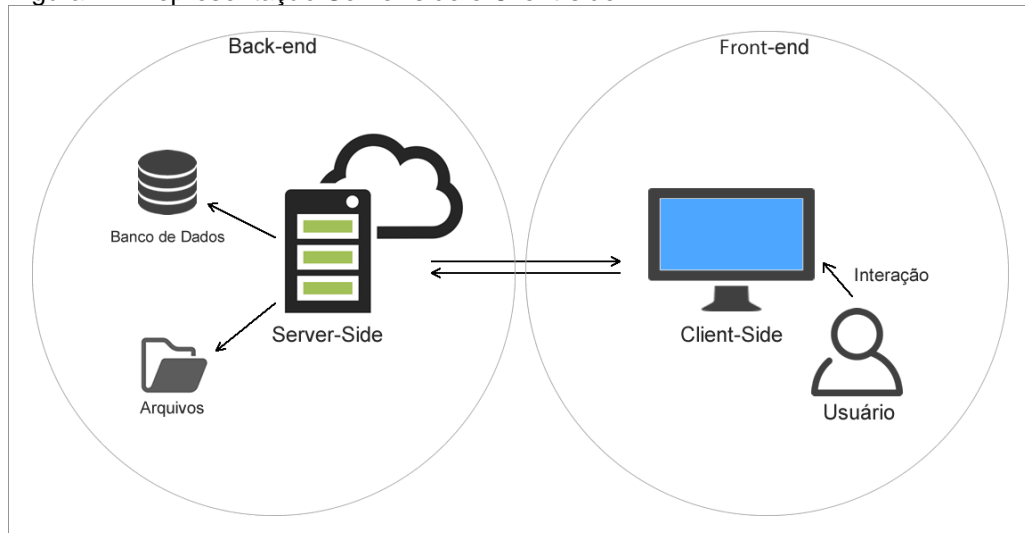
qualquer linguagem, pois utiliza convenções que são familiares às linguagens C, C++, C#, Java, JavaScript, entre outras.

Estas propriedades fazem com que JSON seja um formato ideal de troca de dados. JSON pode representar quatro tipos de dados primitivos (*strings*, *numbers*, *booleans* e *null*) e 2 tipos de estruturas (*object* e *arrays*). Uma *string* é uma sequência de zero ou mais caracteres Unicode. Um *object* é uma coleção desordenada de zero ou mais pares de nome/valor, onde nome é uma *string* e valor pode ser uma *string*, *number*, *boolean*, *null*, *object* ou *array*. Um *array* é uma sequência de zero ou mais valores ordenados. O termo *object* e *array* vem das convenções de JavaScript.

2.7 DESENVOLVIMENTO PARA A WEB

O desenvolvimento para a Web envolve desde marcação de texto até POO. Diversas linguagens e tecnologias podem ser utilizadas para a construção de uma página Web. Um sistema Web pode ser dividido em dois lados, o lado do cliente (*client-side*) e o lado do servidor (*server-side*).

Figura 1 – Representação Server-side e Client-side



Fonte: Do autor.

O termo *front-end* e *back-end* são usados para caracterizar as interfaces e serviços do programa de acordo com o lado em que estão servindo.

2.7.1 Front-end

Em sistemas Web, um aplicativo *front-end* é aquele com o qual os usuários do aplicativo interagem diretamente com a aplicação (SAHOO; SAHOO, 2016, tradução nossa). Aplicativos *front-end* são executados no navegador (*client-side*), sendo assim, seus códigos são visíveis aos usuários. Um aplicativo de *front-end* pode encaminhar solicitações para um programa de *back-end* localizado em outro computador para obter dados solicitados ou executar um serviço solicitado. O *front-end* é separado em 3 camadas básicas, a camada de informação (HTML), a camada de formatação (CSS), e a camada de comportamento (Javascript). São algumas das tecnologias que se destacam no desenvolvimento *front-end* o: *HTML5*, *CSS3*, *Bootstrap*, *Materialize*, *Javascript*, *Angular*, *Jquery*, dentre outros.

2.7.1.1 HTML5

A linguagem HTML é uma linguagem de marcação de texto através do uso de tags, sendo uma recomendação da *World Wide Web Consortium (W3C)*. As tags descrevem a estrutura de uma página, desde a marcação de textos até a geração de gráficos (W3C, 2017, tradução nossa). A versão mais recente, HTML5 atribui semântica a algumas de suas tags, permitindo a construção de páginas estruturada somente com o uso de tags HTML. Uma tag é definida entre os caracteres < e >, sendo que o nome da tag deve ser informado logo após o caracter <. Atributos e conteúdo podem ser adicionados a uma tag, os atributos são descritos após o nome da tag, sempre com um *espaço* antes de sua declaração, enquanto que o conteúdo é inserido após a declaração da tag e seguido de uma tag de fechamento, que corresponde ao mesmo nome da tag de abertura com adição do prefixo /. As tags *area*, *base*, *br*, *col*, *embed*, *hr*, *img*, *input*, *link*, *meta*, *param*, *source*, *track* e *wbr* são consideradas vazias, pois não possuem conteúdo tão pouco precisam ser fechadas (W3C, 2017, tradução nossa).

Figura 2 – Exemplos de tags HTML.

```

1
2 <!-- Tag de Comentário -->
3
4 <!--
5     Tag com nome 'span' abaixo
6     a tag span possui um atributo 'id' cujo valor é 'meu_id'
7     o conteúdo ' texto qualquer ' esta presenta nesta tag span
8 -->
9 <span id='meu_id'> texto qualquer </span>
10
11 <!--
12     Tag com nome 'img' abaixo
13     atributo 'src' fundamental para o funcionamento desta tag
14     o valor do atributo src é 'img/imagem.png'
15     a tag img é um exemplo de tag vazia, por isso não é fechada
16 -->
17 <img src='img/imagem.png'>

```

Fonte: Do autor.

Dentre algumas das regras que definem a estrutura das tags HTML, as mais relevantes determinam que todas as tags e seus atributos devem ser escritos em letras minúsculas, e os valores dos atributos devem estar entre aspas (W3C, 2017, tradução nossa).

2.7.1.2 CSS3

Cascading Style Sheets (CSS) é uma linguagem utilizada para adicionar estilos a uma página Web através de suas tags, separando o conteúdo de um documento da sua formatação (W3C, 2017, tradução nossa). A versão mais atual, a CSS3 traz animações e efeitos para serem aplicados as tags HTML.

A sintaxe utilizada nos arquivos com formato CSS é simples e baseada em uma lista de regras. Cada navegador possui predefinições de estilos aplicadas a cada tag HTML, sendo assim, uma página Web pode apresentar estilos e formatações diferentes de navegador para navegador (W3C, 2017, tradução nossa). Para resolver este problema e garantir uma normalização da página entre os navegadores, uma predefinição em css deve ser incluída, garantindo assim a substituição das definições dos navegadores pelas que foram aplicadas. Dentre as opções que estão disponíveis gratuitamente na web para a solução deste problema, uma muito utilizada é a *normalize.css*.

2.7.1.3 JavaScript

JavaScript (JS) é uma linguagem de programação *client-side* Orientada a Objetos criada por Brendan Eich em 1995 e mantida pela ECMA desde 1997 (ECMA INTERNATIONAL, 2013, tradução nossa). O nome oficial do padrão é ECMA-262, e ECMAScript é o nome oficial da linguagem. Apesar do nome semelhante, JavaScript não tem relação com a linguagem Java.

JavaScript é uma linguagem leve, interpretada e baseada em objetos, utilizada para manipular o comportamento de uma página através do controle do HTML e CSS (ECMA INTERNATIONAL, 2013, tradução nossa). Diversas bibliotecas foram desenvolvidas em cima do JavaScript para agilizar o desenvolvimento. Alguns dessas bibliotecas são o *React*, *Jquery*, *Angular*, *Lodash*, *Vue*, *Ionic*, *Chart*, entre outros.

2.7.2 Back-end

Um aplicativo ou programa *back-end* serve indiretamente em suporte aos serviços de *front-end*, residindo no servidor da aplicação e podendo interagir diretamente com o *front-end* (SMITH, 2012, tradução nossa).

Um aplicativo *back-end* tem a capacidade de se comunicar com recursos de armazenamento persistente e controle de arquivos do sistema. *Java*, *JSP*, *PHP*, *Python*, *Node.js*, *Ruby*, *ASP* são algumas das tecnologias utilizadas no desenvolvimento *back-end*.

2.8 TRABALHOS CORRELATOS

Reserva-se este capítulo para a apresentação de dois trabalhos correlatos, cujo qual abordam o mesmo objetivo deste trabalho, o uso de metadados para geração de UI para operações CRUD. Porém com o uso de tecnologias e meios diferentes para se alcançar.

O trabalho *Geração Automática e Assistida de Interfaces de Usuário* foi apresentado pelo Mrack (2009) para conclusão da pós-graduação em computação na Universidade Federal do Rio Grande do Sul. O software desenvolvido na linguagem *Java* chamado de MERLIN tem como finalidade gerar automaticamente e

em tempo de execução, interfaces de usuário para operações CRUD baseadas em modelos.

Este trabalho apresenta semelhanças em quase todos os aspectos em comparação ao trabalho do Mrack (2009), ao qual, não faz uso de dicionário de metadados como fonte de variáveis para a geração das UI para operações CRUD.

Em conclusão, o trabalho do Mrack (2009) declara que sua solução apresenta um modelo elegante e com bom nível de abstração para o desenvolvimento de interfaces de usuário, mas que se tornam inviáveis para a maioria dos desenvolvedores devido a frequente utilização de linguagens proprietárias e sintaxes de difícil compreensão.

O trabalho *Geração de Operações CRUD a partir de Metadados* foi apresentado por Arend (2011) na cidade de Santa Maria para o grau de bacharel em ciência da computação na Universidade Federal de Santa Maria. O trabalho tem como objetivo automatizar a criação de páginas *web* para operações CRUD desenvolvidas com *JavaServer Faces*. Utilizando o motor de templates *Apache Velocity* juntamente com templates predefinidos, e usando metadados obtidos do SGBD como variáveis de entrada.

A proposta deste trabalho tem o mesmo objetivo proposto por Arend (2011), porém utilizando tecnologias diferentes para o seu desenvolvimento.

Arend (2011) concluiu que o objetivo do trabalho foi alcançado, apesar de não apresentar uma interface e de precisar de melhorias no processo de MOR, onde teve grandes dificuldades.

Em ambos os trabalhos aqui analisados, a geração de UI para operações CRUD não oferece suporte para relacionamento entre entidades, sendo que ambos recomendam a implementação deste recurso para trabalhos futuros. O trabalho aqui proposto, oferece suporte a relacionamento entre entidades como diferencial.

3 GERAÇÃO DE UI PARA OPERAÇÕES CRUD

Este capítulo apresenta a ferramenta para geração de operações CRUD a partir de metadados. Nas Seções subsequentes, descreve-se o funcionamento de forma geral, seguido de uma abordagem mais detalhada, e por último suas configurações, metadados, tipos de dados e templates utilizados.

3.1 VISÃO GERAL

Em sistemas web que utilizam banco de dados, é saliente a presença das interfaces de usuários para manipulação das informações provenientes do SGBD, as quais são conhecidas como UI para operações CRUD.

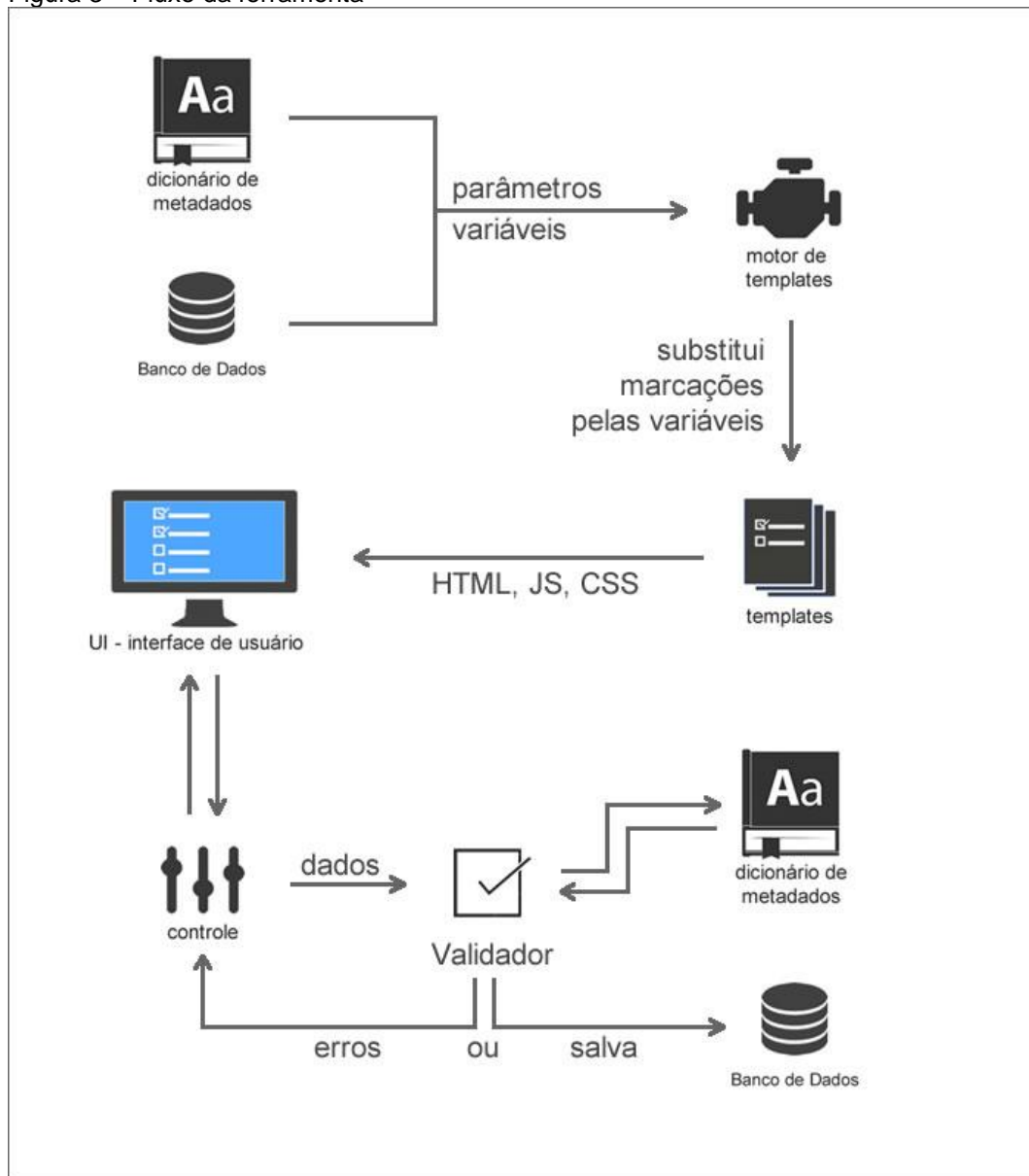
Embora muitas dessas interfaces sejam simplistas, acabam consumindo tempo considerável no desenvolvimento de sistemas web. Segundo Mrack (2007), com base em uma pesquisa realizada em quinze sistemas de uso corporativo, cerca de 30% do tempo de desenvolvimento dos sistemas são dedicadas as telas de cadastro.

Com base nesses valores, a proposta da ferramenta é automatizar completamente a geração das interfaces de usuários para operações CRUD. Para abranger a automação de todas as telas de cadastros, configurações adicionais devem ser informadas e eventualmente, devido a derivada necessidade do mercado, novos algoritmos deverão ser acrescentados a ferramenta.

3.2 FUNCIONAMENTO DA FERRAMENTA

A ferramenta foi desenvolvida em PHP, linguagem back-end largamente utilizada em desenvolvimento de softwares livres e de fácil utilização. A chamada da ferramenta deve ser feita na linguagem PHP, passando como parâmetro um dicionário de metadados. Os dicionários de metadados representam as entidades do sistema, armazenando todos os parâmetros que a ferramenta necessita para a geração da UI para operações CRUD. A estrutura retornada pela ferramenta não é de fato gerada, todos os códigos são previamente desenvolvidos no formato de templates, e o dicionário de metadados filtra os códigos, moldando e retornando apenas os códigos necessários.

Figura 3 – Fluxo da ferramenta



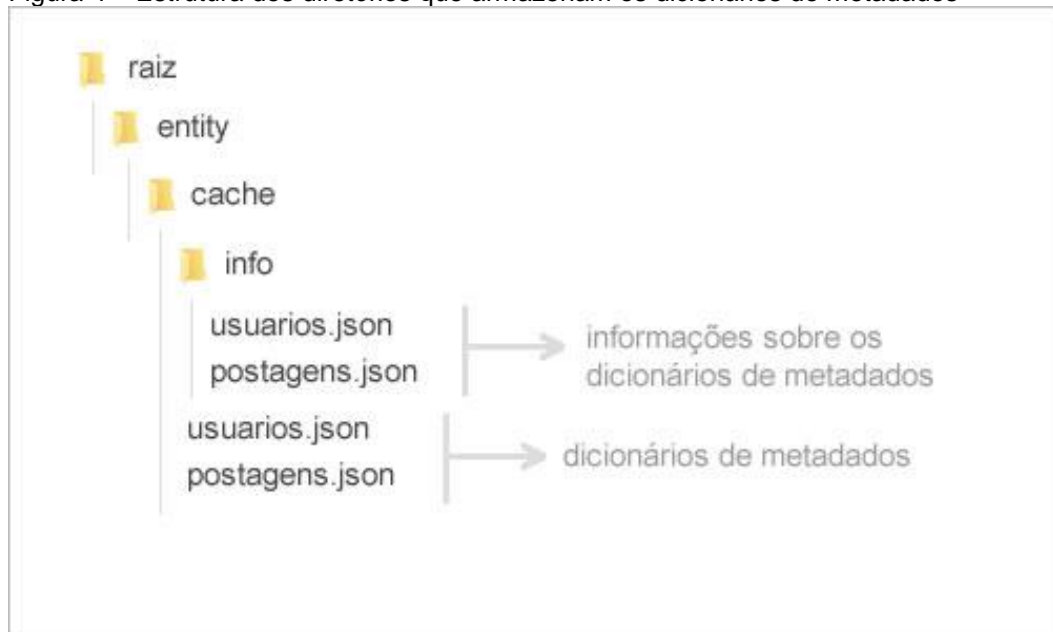
Fonte: Do autor.

3.2 Configurações Iniciais

A conexão com o SGBD Mysql é a primeira necessidade presente em qualquer sistema web que necessite armazenar dados de forma persistente. Com a conexão ao banco de dados estabelecida, é necessário criar os dicionários de metadados, cada entidade presente no sistema necessita de um, pois as tabelas e relações presentes no banco de dados são criadas através das informações presentes no dicionário de metadados. Todas as informações referentes as entidades são armazenadas no diretório *entity* localizado na raiz do projeto, dentro

do diretório *entity*, outro diretório com o nome *cache* irá manter todos os dicionários em formato *json*, e dentro do diretório *cache* um outro diretório de nome *info* irá ser gerada pela ferramenta, mantendo informações sobre os dicionários de metadados, a estrutura pode ser vista abaixo na Figura 4.

Figura 4 – Estrutura dos diretórios que armazenam os dicionários de metadados



Fonte: Do autor.

3.3 Dicionário de Metadados

Os dicionários de metadados são arquivo *json* mantidos no diretório *entity/cache*. Dicionários de metadados são constituídos por listas de pares de chave/valor, a primeira lista presente no dicionário representa os campos da entidade, sendo a chave um identificador numérico único do campo, e o valor uma lista de atributos no formato chave/valor. Cada campo representa uma informação a ser armazenada no SGBD, e seus atributos determinam o formato do dado a ser armazenado, além de delimitar e restringir os valores aceitos pelo campo. Os atributos controlam por exemplo, o estilo do campo na interface de usuário, seu nome, tipo de dado, valor padrão, se permite valor vazio, etc.

Figura 5 – Modelo de Dicionário de metadados

```

1 {
2   "1": {
3     "format": "title",
4     "type": "varchar",
5     "nome": "Nome",
6     "column": "nome",
7     "size": "127",
8     "key": "title",
9     "unique": "true",
10    "default": "false",
11    "update": "true",
12    "relation": "",
13    "allow": {
14      "regex": "",
15      "validate": ""
16    },
17    "form": {
18      "input": "text",
19      "cols": "12",
20      "col": "",
21      "coll": "",
22      "class": "",
23      "style": "font-size:1.7em;font-weight:bold"
24    }
25  },
26  "2": {
27    "format": "link",
28    "type": "varchar",

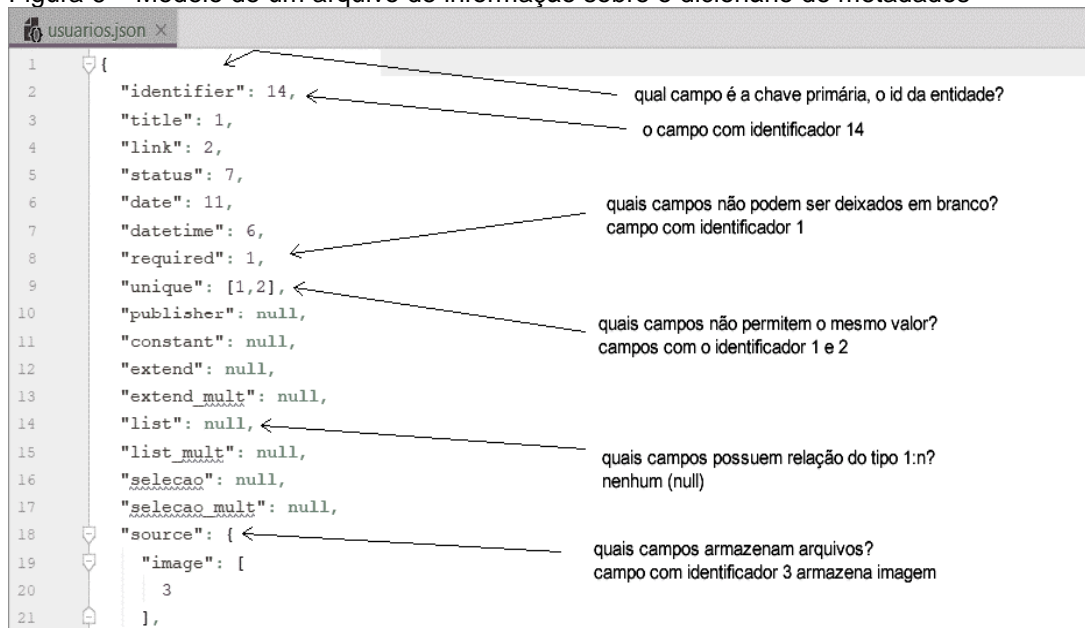
```

Fonte: Do autor.

3.3.1 Informações Sobre os Dicionários de Metadados

As informações sobre os dicionários de metadados ficam localizado no diretório *entity/cache/info* e são arquivos automaticamente gerados pelo sistema com o objetivo de melhorar a performance da ferramenta. Estes arquivos são gerados sempre que o dicionário de metadados é carregado para o sistema e o mesmo não encontra um arquivo de informação relacionado. Os arquivos de informação armazenam dados sobre os dicionários de metadados, para que seja desnecessário a busca por informação em tempo de execução. Por exemplo, é possível buscar informações sobre quais campos possuem relações e com quem, assim como é possível identificar quais campos são únicos, ou quais campos não podem ser deixados em branco, ou até mesmo, identificar a função de um campo na entidade, como a chave primária por exemplo.

Figura 6 – Modelo de um arquivo de informação sobre o dicionário de metadados



Fonte: Do autor.

3.4 Banco de Dados

O SGBDR utilizado pela ferramenta é o Mysql 5.7.14, com suporte a dados do tipo *json*, formato cujo qual é adotado pela ferramenta para armazenar determinados tipos de campo. Caso a versão do Mysql não possua suporte a dados do tipo *json*, o mesmo será tratado como um dado *varchar*. Sempre que o sistema gera o arquivo de informação, uma verificação é realizada no banco de dados para identificar se a tabela com o mesmo nome da entidade existe. Caso não exista, a tabela e seus respectivos campos são criados de acordo com os dados presentes no dicionário de metadados. Em campos relacionais, caso a tabela relacional não exista, a ferramenta busca pelo dicionário de metadados desta entidade relacional para adicioná-la primeiro ao banco de dados e posteriormente seguir com a criação da entidade atual. Caso a tabela já exista, uma verificação é realizada para identificar a existência de alterações nos campos da tabela. Alterações nos campos de uma entidade no banco de dados que já possuem dados armazenados, podem gerar erros de alteração ou erros nos dados armazenados, caso as novas configurações dos campos entrem em conflito com os dados já existentes.

A configuração das tabelas e de seus campos, assim como suas relações no banco de dados são totalmente gerenciadas pela ferramenta tendo como base as informações presentes no dicionário de metadados. Outras funções do banco de

dados Mysql como *Views*, *triggers* e *procedures*, não são abordadas pela ferramenta, sendo assim, caso tenha a necessidade de seu uso, o mesmo deverá ser configurado manualmente. Qualquer alteração manual realizada nas tabelas do banco de dados poderá resultar no mal funcionamento da ferramenta.

3.5 Tipos de Dados

Os tipos de dados definem todos os diferentes tipos de dados que poderá ser utilizada nos campos de uma entidade, esses tipos são declarados em um arquivo *json* localizado no diretório *entity* na raiz do sistema com nome *input_type*. Além de definir os tipos de dados aceito, o *input_type* também define os atributos padrões para os campos de cada tipo de dado, ou seja, caso seja omitido pelo dicionário de metadados, esses valores serão utilizados no lugar. Os tipos de dados variam de sistema para sistema, e podem ser reaproveitáveis ao longo das implementações.

Para a criação de um tipo de dado, é necessário que o mesmo seja definido no arquivo *input_type*, além disso, é necessário um template com o mesmo nome do tipo, para que este seja utilizado na UI. Por último, é preciso configurar a classe responsável pela criação dos campos no SGBD, para que o tipo de dado seja criado no banco de dados de forma a corresponder o seu valor. Dentre as necessidades encontradas para o desenvolvimento deste trabalho, diversos tipos de dados foram desenvolvidos. Algumas semelhanças entre os tipos de dados foram identificadas e separadas em quatro classificações de tipos de dados diferentes.

A primeira classificação, é a simples, dados genéricos constituídos por caracteres, números, datas, etc. O tipo de dado *source*, é um exemplo da classificação simples, e também o mais diferente entre o seu tipo, pois permite carregar arquivos para o sistema, além de armazenar dados no SGBD, alguns exemplos de dados do tipo *source* são imagens, áudios e documentos.

Após, temos a classificação semântica, que são dados simples, porém com funções predefinidas para o tipo de dado, ou seja, um tipo de dado semântico, sempre terá a mesma função de entidade para entidade, são exemplos de dados semânticos, os tipos *cep*, *cpf*, *título*, *telefone*, etc.

A terceira classificação é a relacional, dados que estabelecem conexões entre entidades, estes dados são valores inteiros que armazenam o *id* do dado da entidade relacionada.

Por último, temos a classificação informativa, um tipo de dado especial, pois não tem como objetivo armazenar dados, este tipo de dado não é gerado no SGBD e serve apenas para adicionar conteúdo HTML ao formulário, como textos, descrições, imagens, etc.

3.5.1 Dados Relacionais

Os dados relacionais são dados que permitem estabelecer relações entre dados de duas ou mais entidades, estas relações podem se dar de diferentes formas e comportamentos, para distinguir suas diferenças, sete tipos de dados relacionais foram adicionados ao arquivo *input_type*, sendo assim aceitos pelo dicionário de metadados. Os tipos são, *publisher*, *extend*, *list*, *selecao*, *extend_mult*, *list_mult*, *selecao_mult*. Os quatro primeiros tipos de dados são referentes as relações únicas entre a entidade e a entidade relacional, enquanto que as três últimas se referem a múltiplas relações de dados entre as entidades. O tipo *list* e o tipo *selecao*, assim como o tipo *list_mult* e o tipo *selecao_mult* são idênticos em sua apresentação no SGBD, porém apresentam funções diferentes na UI. Para os tipos de dados *list* e *list_mult* é possível tanto selecionar quanto adicionar dados à entidade relacional, enquanto que a *selecao* e *selecao_mult*, permite apenas selecionar os dados. Tendo as diferenças entre *list* e *selecao*, assim como *list_mult* e *selecao_mult* esclarecidas, sempre que for citado o tipo de dado *list* ou *list_mult*, considere aplicável também ao tipo *selecao* e *selecao_mult*.

A diferença entre os tipos de dados *extend* e *list*, está no fato de que as informações armazenadas na tabela relacional *list* são reutilizadas por outras entidades, permitindo que diversas entidades referenciem o mesmo valor, são exemplos comuns de uso da relação *list* em entidades que representam categorias, tags, ceps, etc. Enquanto que o tipo *extend* armazena valores não compartilhados, ou seja, de uso exclusivo da entidade que o relacionou, são exemplos de uso em entidades que representam contatos, endereços, etc.

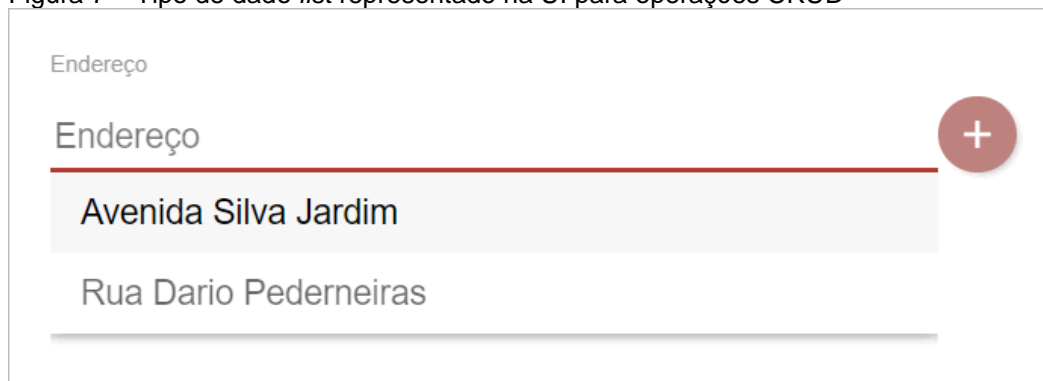
O tipo de dado *extend* possui uma representação única na UI, pois todos os campos da entidade relacional são incorporados no formulário principal da

entidade, não apresentando diferença na UI entre os dados da entidade e da entidade relacional. Porém, ao armazenar os dados, cada entidade armazena os seus dados de forma independente com uma relação estabelecida entre ambas.

O tipo de dado *extend_mult* permite criar uma lista de dados únicos relacionados a entidade. Um exemplo do uso de tipo de dado *extend_mult* se dá, quando se tem mais de um endereço.

O tipo *list* gera um campo do tipo autocomplete na UI, juntamente com um botão para adicionar nova entrada, que permite selecionar um dado já existente na entidade relacional, ou então, adicionar um novo dado. Na figura 7 abaixo, é possível verificar o modelo do tipo de dados *list* na UI para operações CRUD gerada pela ferramenta.

Figura 7 – Tipo de dado *list* representado na UI para operações CRUD



Endereço

Endereço

Avenida Silva Jardim

Rua Dario Pederneiras

Fonte: Do autor.

O tipo *list_mult* é semelhante ao *list* tanto em seu armazenamento quanto na apresentação na UI, porém permite adicionar ou selecionar diversos dados da entidade relacional, gerando uma lista de dados relacionados logo abaixo do campo.

Por último, temos o tipo de dado *publisher*, um dado utilizado para relacionar o dado armazenado na entidade a um usuário do sistema. Desta forma é possível adicionar um autor ao conteúdo, permitindo que apenas o autor ou usuário superior tenha acesso a alterações. Neste tipo de dado, é necessário que o sistema possua controle de acesso de usuários, com nível de acesso, e uma entidade para o representar. No desenvolvimento deste trabalho, é utilizado como base a entidade *usuarios* como sendo a entidade de acesso ao sistema.

3.6 Templates

Para que os templates sejam utilizados, um motor de templates precisa processá-los, o *Smarty* é um motor de templates leve, simples e de fácil implementação, que considera todos os tipos de arquivos com extensão *tpl* como templates. Todos os templates utilizados pela ferramenta estão localizadas na raiz do sistema, no diretório *tpl*. Um template deve ser escrito da mesma forma que um documento HTML. Para cada tipo de dado existente, visto na Seção 3.5, um template com mesmo nome deve estar presente no diretório *tpl*.

Na Seção 3.2 foi visto que a UI para operações CRUD retornada é um modelo previamente desenvolvido, não sendo gerada de fato, isso porque, a ferramenta retorna uma combinação de templates alimentada com variáveis do dicionário de metadados. Para isso, a ferramenta percorre todos os campos do dicionário de metadados, associando um template para cada campo, fazendo a substituição das marcações destes templates pelos atributos presentes no dicionário. Ao final, um formulário com os campos necessários será montado pelo motor de templates, e retornado ao usuário.

3.7 Controladores

A ferramenta pode retornar a interface de usuário para operações CRUD com valores já presentes nos campos, neste caso, é dito que a UI está em modo de edição. Para isso, é necessário passar o id (chave primária) do dado que deseja editar na chamada da ferramenta. Ao retornar a UI para operações CRUD em uma página web, os controladores entram em ação, basicamente a função dos controladores é detectar alterações e reagir as ações no formulário, buscando informações, suprimindo o formulário, validando dados e submetendo as informações para o SGBD.

Ao editar algum campo da UI para operações CRUD, o formulário é submetido para revisão na ferramenta, onde através da consulta ao dicionário de metadados, a ferramenta valida cada campo submetido, verificando se o valor informado condiz com os atributos do dicionário de metadados. Caso todos os dados sejam válidos e os dados necessários estão presentes, a ferramenta salva os dados no banco de dados e retorna para o controlador o id do dado salvo. Assim que o

controlador identifica o id retornado, o mesmo entra em modo de edição, caso não estivesse.

3.8 Personalização do Formulário

A fim de atender algumas necessidades presentes no desenvolvimento de UI para operações CRUD, devido as derivadas necessidades do mercado, configurações extras podem ser aplicadas a ferramenta. Entre elas, é possível configurar o formato do botão de salvar, delimitar os campos que serão exibidos no formulário e disparar uma ação ao salvar o formulário. Além das configurações extras aplicadas na ferramenta, também é possível aplicar configurações extras nos campos relacionais, delimitando os campos que serão necessários ou filtrando os campos a serem selecionados.

4 APRESENTAÇÃO E ANÁLISE DOS DADOS

Este capítulo destina-se à aplicação dos conceitos apresentados nos capítulos anteriores, utilizando como meio, para demonstração do uso da ferramenta para geração automatizada de UI para operações CRUD, um exemplo aplicado a uma empresa de pequeno porte na área de jornalismo.

4.1 CONFIGURAÇÕES INICIAIS

Antes de iniciar o desenvolvimento de um sistema web, é necessário que os recursos e softwares utilizados no projeto estejam presentes. Para satisfazer as necessidades da ferramenta aqui proposta, o software *WampServer* para Windows 10 foi utilizado, ao qual permite criar aplicativos web com apache2, PHP e MySQL. Com o WampServer instalado e em execução, o sistema dispõe de todos os recursos necessários para rodar o projeto sem a necessidade de conexão com a internet. Antes de começar o desenvolvimento, é preciso criar a base de dados (*database*) no SGBD e estabelecer a conexão com a aplicação.

4.1 CENÁRIO

Para demonstrar os benefícios do uso da ferramenta, este exemplo fez uso de diversos tipos de dados, a fim de demonstrar alguns dos recursos que a ferramenta oferece. Para isso, foi utilizado como exemplo, uma situação real de uma pequena empresa de jornalismo que deseja publicar suas notícias online em um sistema web.

Neste contexto, será necessária uma entidade para controle de acesso ao sistema, que identificará o usuário e suas permissões. Para a publicação de notícias, é preciso uma entidade para armazenar as notícias publicadas, identificando o autor da notícia através da entidade usuários. Por fim, para adicionar categorias as notícias a fim de separá-las por área, outra entidade será utilizada para armazenar todas as categorias do sistema, ao qual as notícias poderão se relacionar. Na figura 8 abaixo, é possível verificar a estrutura das entidades citadas acima.

Figura 8 – Entidades necessárias para o exemplo da empresa de jornalismo

Nome da Entidade	Nome da Entidade	Nome da Entidade
usuarios	noticias	categorias
id	id	id
nome	titulo	nome
nome_usuario	name	link
imagem	conteudo	descricao
email	imagem	data
password	data	
data	autor	
status	categorias	
nivel		
token		

Fonte: Do autor.

A fim de centrar o foco das atividades desenvolvidas nesse exemplo na ferramenta proposta, outras atividades complementares envolvidas no exemplo não serão abordadas, como o controle de acesso dos usuários, e do uso de ferramentas terceirizadas.

4.2 TIPOS DE DADOS UTILIZADOS

O campo *autor* presente na entidade *noticias* é um campo relacional do tipo de dado *publisher*, onde armazenará o *id* do usuário que criou ou modificou os dados salvos em *noticias*. Além do campo *autor*, *noticias* também possui um campo relacional do tipo *list_mult*, onde permite que a entidade receba diversos dados relacionados a entidade *categorias*.

De forma geral, os campos *id*, são campos semânticos utilizados para determinar um identificador único para cada dado armazenado, sendo a chave primária de cada registro. Os campos *nome* e *titulo* também são campos semânticos do tipo *titulo*, utilizados para identificação de registro por uso de caracteres alfanumérico. Os campos *nome_usuario*, *name* e *link* são dados semânticos do tipo *link*, que por padrão, não são mostrados na UI, mas armazenam uma versão sem

espaço do título. Os campos *imagem*, são dados primários do tipo *source*, e armazenam os dados sobre uma imagem carregada para o sistema. Os campos *data* são dados semânticos do tipo *date_time*, que por padrão, não são mostrados na UI, armazenando a data e hora de criação do registro. Os campos *conteudo* e *descricao*, são dados primários do tipo *html* e *textarea* respectivamente, que são campos para adição de conteúdo descritivo. Os demais campos *email*, *password*, *status*, *nível* e *token* presentes na entidade *usuarios*, são campos para controle de acesso.

4.3 DESENVOLVIMENTO DOS DICIONÁRIOS DE METADADOS

Antes de criar os dicionários de metadados, é preciso que todos os tipos de dados estejam presentes no sistema. No geral, todos os tipos de dados precisam estar definidos no arquivo *input_type*, com os valores padrões a serem aplicados caso estes sejam omitidos no dicionário de metadados, além é claro, de um template associado para exibição na UI, conforme especificado na seção 3.5. O arquivo *input_type* se encontra no apêndice A, e pode ser reaproveitável em futuros sistemas. Os templates referentes a cada tipo de dado são encontrados logo após, no apêndice B.

Após identificar as entidades necessárias para o desenvolvimento do sistema, e ter em mãos os tipos de dados e campos necessários, basta desenvolver os dicionários de metadados de cada entidade. Para visualizar os dicionários de metadados desenvolvidos neste exemplo, ide ao apêndice C.

4.4 GERANDO A UI PARA OPERAÇÕES CRUD

Uma vez que os dicionários de metadados foram definidos, a ferramenta pode ser iniciada através de um arquivo PHP de acordo com a figura 9.

Figura 9 – Chamada da ferramenta em um arquivo PHP

```

1  <?php
2
3  echo "<div class='container-900 padding-64'>" .
4      "<div class='col card padding-medium color-white'>";
5
6  $form = new Form("noticias");
7  $form->showForm();
8
9  echo "</div></div>";
10

```

Fonte: Do autor.

A função executada na figura 9 buscou o dicionário de metadados com o nome de arquivo *noticias.json* localizado no diretório *entity/cache* para geração da UI para operações CRUD. No processo de geração da interface, a ferramenta gerou as tabelas no SGBD para a entidade alvo e suas relações, conforme visto nas figuras 10, 11 e 12.

Figura 10 – Tabela *noticias* gerada no SGBD MySQL pela ferramenta

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	None		AUTO_INCREMENT
2	titulo	varchar(127)	utf8_general_ci		No	None		
3	name	varchar(127)	utf8_general_ci		No	None		
4	conteudo	text	utf8_general_ci		Yes	NULL		
5	imagem	json			Yes	NULL		
6	data	datetime			Yes	NULL		
7	autor	int(11)			Yes	NULL		

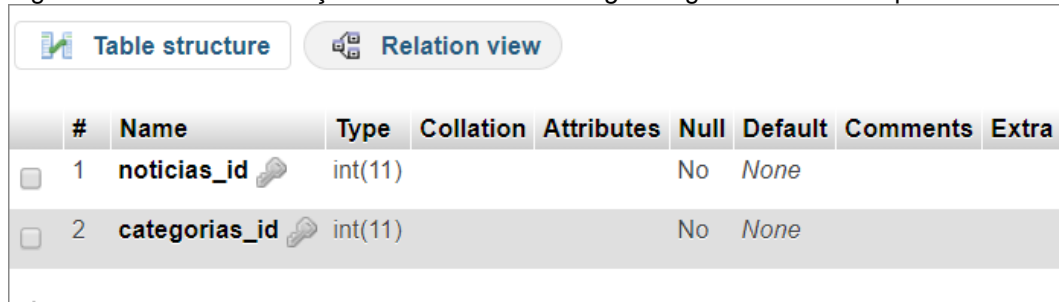
Fonte: Do autor.

Figura 11 – Tabela *categorias* gerada no SGBD MySQL pela ferramenta

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	None		AUTO_INCREMENT
2	nome	varchar(127)	utf8_general_ci		No	None		
3	link	varchar(127)	utf8_general_ci		No	None		
4	descricao	varchar(1023)	utf8_general_ci		Yes	NULL		
5	data	datetime			Yes	NULL		

Fonte: Do autor.

Figura 12 - Tabela de relação entre *noticias* e *categorias* gerada no SGBD pela ferramenta



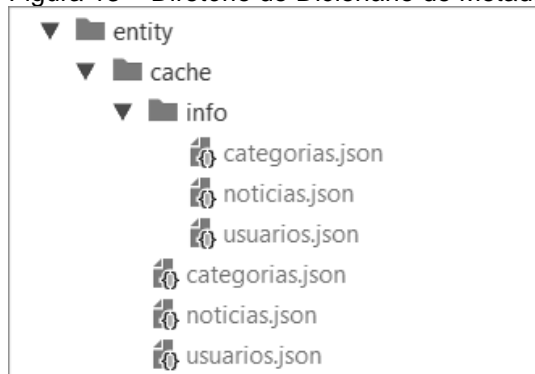
The screenshot shows a window with two tabs: 'Table structure' (selected) and 'Relation view'. Below the tabs is a table with the following structure:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	noticias_id	int(11)			No	None		
2	categorias_id	int(11)			No	None		

Fonte: Do autor.

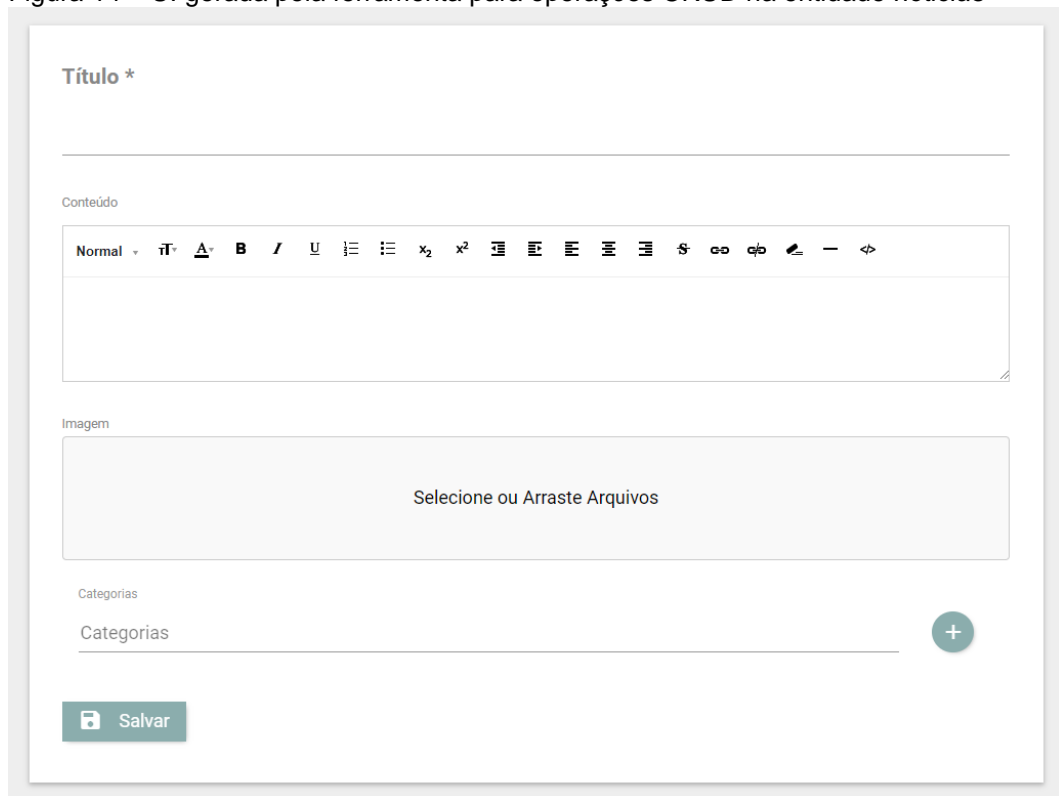
Além das tabelas no SGBD, a ferramenta também criou os arquivos de informações das entidades localizados no diretório *entity/cache/info*, ao qual servem para melhoria na performance da ferramenta, não interessando para o usuário da ferramenta. Neste exemplo, o diretório *entity* ficou conforme figura 13.

Figura 13 – Diretório do Dicionário de Metadados



Fonte: Do autor.

Por fim, temos a UI para operações CRUD retornada pela ferramenta, com todos os controladores e estilos necessários para o seu funcionamento, conforme figura 14.

Figura 14 – UI gerada pela ferramenta para operações CRUD na entidade *noticias*

The image shows a web form for creating or editing a news item. It features the following elements:

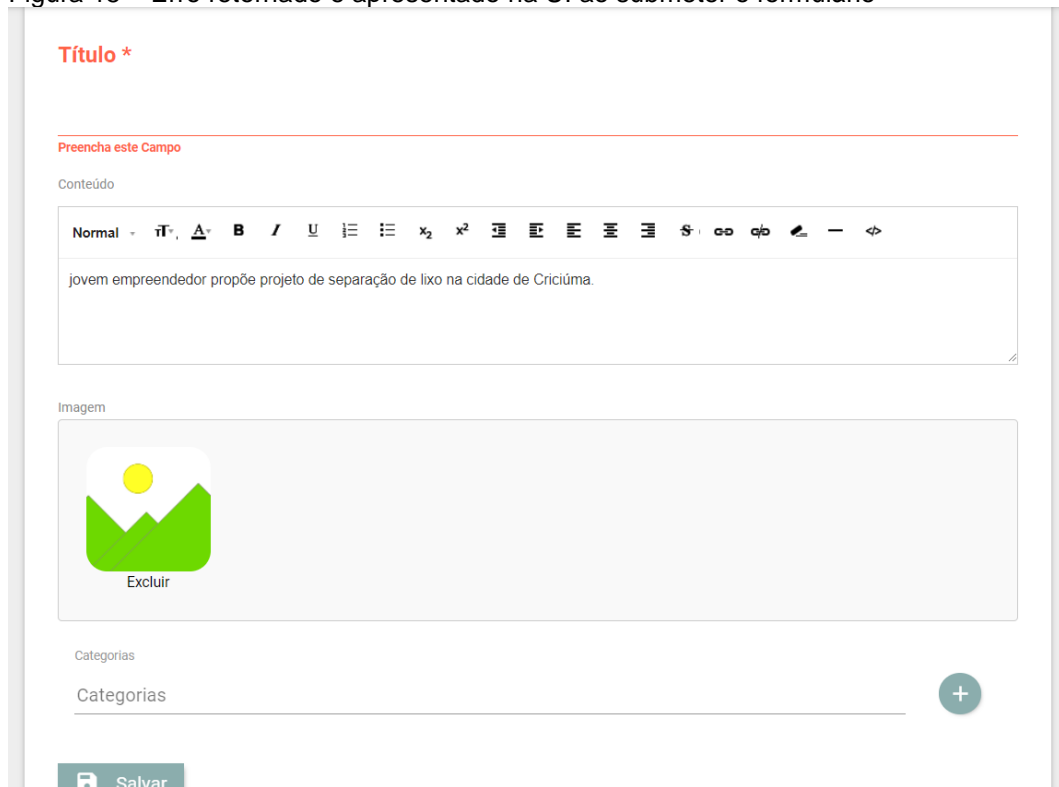
- Título ***: A required text input field.
- Conteúdo**: A rich text editor with a toolbar containing icons for bold, italic, underline, list, link, unlink, and other text formatting options.
- Imagem**: A large rectangular area with the text "Selecione ou Arraste Arquivos" (Select or Drag Files), indicating a file upload feature.
- Categorias**: A text input field with a dropdown arrow and a green plus icon to the right, suggesting a selection of categories.
- Salvar**: A green button with a floppy disk icon and the text "Salvar" (Save).

Fonte: Do autor.

4.5 UTILIZANDO A UI PARA OPERAÇÕES CRUD

A interface de usuário retornada pela ferramenta possui os campos necessário conforme especificado no dicionário de metadados, assim como os controladores e estilos para o correto funcionamento da utilização do cadastro. A fim de verificar se o formulário retorna os erros conforme as regras presentes no dicionário de metadados, o Título, um campo obrigatório, foi deixado em branco e submetido para ser salvo, o retorno foi um erro tratado pelo controlador e exibido na interface, conforme figura 15.

Figura 15 – Erro retornado e apresentado na UI ao submeter o formulário



The image shows a web form with the following elements:

- Título ***: A label for the title field.
- Preencha este Campo**: An error message displayed in red text above the content field.
- Conteúdo**: A rich text editor containing the text "jovem empreendedor propõe projeto de separação de lixo na cidade de Criciúma".
- Imagem**: An image placeholder with a green and yellow icon and the label "Excluir".
- Categorias**: A field with a plus sign button for adding categories.
- Salvar**: A button at the bottom of the form.

Fonte: Do autor.

Após preencher o campo Título com um valor qualquer válido, novamente foi submetido para ser salvo, o retorno desta vez foi uma mensagem informando o sucesso da ação e limpando o formulário para nova entrada de registro, conforme figura 16. Esta ação de retorno é padrão e pode ser alterada passando configurações extras na chamada da ferramenta.

Figura 16 – Cadastro de registro realizado com sucesso na entidade *noticias*

The screenshot shows a registration form for news items. It features a title field, a rich text editor for content, an image upload area, and a category selection field. A success message 'Cadastro Salvo!' is displayed in a green box, and a 'Salvar' button is located at the bottom left of the form.

Fonte: Do autor.

Figura 17 – Registro inserido no SGBD na tabela *noticias* após cadastro submetido

id	titulo	name	conteudo	imagem	data	autor
1	Jovem Empreendedor	jovem-empresendedor	jovem empreendedor propõe projeto de separação de ...	[[{"url": "uploads\\image\\2018\\06\\demo.png", "na..."}]]	2018-06-11 19:43:10	1

Selected: Edit Copy Delete Export

Fonte: Do autor.

4.6 EDITANDO E EXLUINDO REGISTROS

Para editar um registro armazenado no SGBD, é necessário passar o seu *id* como parâmetro na chamada do método *showForm*, conforme figura 18. Ao chamar uma UI com um *id* válido como parâmetro, a ferramenta entra em modo de edição, sendo assim, algumas ações tomam validações diferentes, além de incluir um botão para excluir o registro no final do formulário, conforme visto na figura 19.

4.7 RESULTADOS OBTIDOS

O exemplo apresentado não envolve necessidades complexas e pode ser considerada um exemplo comum do uso da ferramenta. Foi utilizado diferentes tipos de dados para abordar os modelos de dados distintos que podem ser utilizados pela ferramenta. Com isso, a ferramenta apresentou características importantes, como a aplicação de regras de usabilidade, a validação automática dos dados e máscaras, a aplicação e controle de reponsividade no *layout*.

A ferramenta automatiza a criação de UI para operações CRUD, assim como o controle sobre os dados armazenados, tornando desnecessário qualquer controle sobre as tabelas ou campos presentes no SGBD.

Com o reuso dos tipos de dados aplicados, a solução provoca uma redução na curva de desenvolvimento da ferramenta. Quanto mais modelos e tipos de dados forem adicionados a solução, maiores as possibilidades de uso da ferramenta sem a necessidade de desenvolvimento.

De forma geral, este exemplo demonstra a possibilidade de uso da ferramenta para produção de código-fonte de interfaces de usuários para operações CRUD de forma automatizada.

5 CONCLUSÃO

O trabalho propôs o desenvolvimento de uma ferramenta para a geração automatizada de UI para operações CRUD em sistemas web. Essa geração ocorre com base em dicionários de metadados servindo como variáveis de entrada para um motor de templates.

Com fundamento de reduzir o tempo de desenvolvimento das interfaces de usuários para operações CRUD em sistemas web que representam até 30% do tempo total de desenvolvimento (MRACK, 2007). A ferramenta apresenta uma abordagem diferente no desenvolvimento de UI para operações CRUD, a fim de facilitar e melhorar a flexibilidade da criação de sistemas web.

A geração de UI para operações CRUD mostrou-se complexo ao trabalhar com relacionamentos, pois exige uma combinação de metadados para atender as derivadas necessidades do mercado. A inserção de novos metadados ou tipos de dados, exige a reconfiguração da ferramenta para que possa funcionar em todas as áreas de atuação, sendo elas, a criação das tabelas no SGBD, a criação do arquivo de informação, o mapeamento objeto-relacional e as validações necessárias.

A ferramenta proposta não possui todos os recursos necessários para atender as demasiadas necessidades presentes em sistemas web, sendo necessário a implementação de novos tipos de dados, templates, configurações, controladores e estilos. Porém, uma vez desenvolvido, o código é reaproveitável em futuros sistemas, por isso, o uso deste tipo de ferramenta tende a uma curva decrescente na linha do tempo de desenvolvimento de interfaces de usuários para operações CRUD.

Levando em consideração as dificuldades encontradas, e das limitações que o atual modelo da ferramenta apresenta, o objetivo do trabalho foi alcançado, permitindo a geração automatizada de UI para operações CRUD com base em metadados.

Como sugestão para trabalhos futuros, indica-se a implementação de tipos de dados e recursos na ferramenta desenvolvida, a fim de alcançar a maioria das necessidades presentes em sistemas web. Dentre as dificuldades encontradas, as relações entre entidades se mostrou as mais complexas, devido às inúmeras possibilidades de configuração.

Sugere-se também o desenvolvimento de uma aplicação para a extração de metadados em um banco de dados já existente, a fim de gerar dicionários de metadados a partir de uma base de dados no SGBD. Com isso, a implementação da ferramenta em sistemas já desenvolvidos poderia ser utilizada sem a necessidade de uma readaptação.

Por fim, sugere-se uma interface de usuário para controle e manutenção dos dicionários de metadados, a fim de evitar erros em sua arquitetura e agilizar o desenvolvimento de novos dicionários de metadados.

REFERÊNCIAS

ALVES, Rachel Cristina Vesú. **Metadados como Elementos do Processo de Catalogação**. 2010. 132 f. Tese (doutorado) - Universidade Estadual Paulista, Faculdade de Filosofia e Ciências, 2010. Disponível em: <<http://hdl.handle.net/11449/103361>>. Acesso em: 16 nov. 2017.

AREND, Felipe Gabriel. **Geração de Operações CRUD a partir de Metadados**. 2011. 72 f. TCC (Graduação) - Curso de Curso de Ciência da Computação, Centro de Tecnologia, Universidade Federal de Santa Maria, Santa Maria, 2011. Disponível em: <<http://www-app.inf.ufsm.br/bdtg/arquivo.php?id=153&download=1>>. Acesso em: 24 set. 2017.

BRAY, Tim. **The Javascript Object Notation (JSON) Data Interchange Format**. 2014. Disponível em: <<https://buildbot.tools.ietf.org/pdf/rfc7158.pdf>>. Acesso em: 02 out. 2017.

COELHO, Camila Arnellas; SARTORELLI, Reinaldo Coelho. **Persistência de Objetos via Mapeamento Objeto-Relacional**. 2004. 110 f. TCC (Graduação) - Curso de Sistemas de Informação, Computação e Informática, Universidade Presbiteriana Mackenzie, São Paulo, 2004. Cap. 1. Disponível em: <<http://www.dominiopublico.gov.br/download/texto/ea000253.pdf>>. Acesso em: 15 nov. 2017.

CUER, Eder dos Santos. **Comparação de Desempenho de Bancos de Dados SQL e NoSQL**. 2014. 97 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário Eurípides de Marília, Marília, 2014. Disponível em: <[http://aberto.univem.edu.br/bitstream/handle/11077/999/Eder dos Santos Cuer.pdf?sequence=1](http://aberto.univem.edu.br/bitstream/handle/11077/999/Eder%20dos%20Santos%20Cuer.pdf?sequence=1)>. Acesso em: 15 nov. 2017.

DATE, C. J. **Introdução a Sistemas de Banco de Dados**. Rio de Janeiro: Campus, 2004.

ECMA INTERNATIONAL (Suíça). **The JSON Data Interchange Format**. 2013. Disponível em: <<http://www.json.org/json-pt.html>>. Acesso em: 04 out. 2017.

GARCÍA, F. J.; CASTANEDO, R. I.; FUENTE, A. A. J. **A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications**. 2007. Springer.

LÓSCIO, B. F., OLIVEIRA, H. R. e PONTES, J. C. S. **NoSQL no desenvolvimento de aplicações Web colaborativas**, VIII Simpósio Brasileiro de Sistemas Colaborativos, 2011. Disponível em: <<http://eloquium.com.br/wp/wp-content/uploads/2015/12/PreAulaGrafo4.pdf>>. Acesso em: 20 nov. 2017.

MEDEIROS, Ana Katarina Cavalcanti. **Análise Comparativa Entre Banco De Dados Relacionais E Não Relacionais Com Ênfase No Desempenho De Base De Dados No Contexto De Internet Das Coisas**. 2015. 49 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade de Pernambuco, Caruaru, 2015. Disponível em: <<http://upecaruaru.com.br/wp-content/uploads/2015/07/Ana-Katarina-Cavalcanti-Medeiros.pdf>>. Acesso em: 16 nov. 2017.

MRACK, Marcelo. **Geração Automática e Assistida de Interfaces de Usuário**. 2009. 72 f. TCC (Graduação) - Curso de Computação, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009. Disponível em: <<http://hdl.handle.net/10183/37181>>. Acesso em: 09 maio 2017.

MRACK, M.. **Merlin: Um Novo Horizonte na Criação das Telas de Cadastro**, 8º Fórum Internacional de Software Livre (FISL). Porto Alegre, 2007.

PARR, T. **Enforcing Strict Model-View Separation in Template Engines**. Publicado em: WWW '04: Proceedings of the 13th international conference on World Wide Web. ACM, 2004. Disponível em: <<https://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf>>. Acesso em: 20 nov. 2017.

PROJETANDO BANCOS NOSQL: Conheça algumas técnicas de modelagem utilizadas em bancos NoSQL. Grajaú, Rj: Sql Magazine, v. 117, 2013. Bimestral.

PUERTA, A.; EISENSTEIN, J.; **Towards a General Computational Framework for Model-Based Interface Development Systems**. Stanford University, Califórnia. 1999.

RIBEIRO, Gilberto Pessanha. **Metadados Geoespaciais Digitais**. 1996. 69 f. Tese (Doutorado) - Curso de Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1996. Disponível em: <http://www.geomatica.eng.uerj.br/docentes/gilberto/_media/qualify2_gilberto.pdf>. Acesso em: 20 nov. 2017.

SADALAGE, PRAMOD J.; FOWLER, M. **NoSQL – Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. São Paulo: Novatec, 2013.

SAHOO, Gagan; SAHOO, Reeta. **Infomatic Practices**. Nova Delhi: New Saraswati House, 2016. Disponível em: <<https://www.amazon.com/Infomatic-Practices-Gagan-Sahoo-Reeta-ebook/dp/B01IBGZ4HI>>. Acesso em: 20 nov. 2017.

SHRIVASTAVA, Alok; SOMASUNDARAM. **Armazenamento e Gerenciamento de Informações**: Como armazenar, gerenciar e proteger informações digitais. São Paulo: Bookman Editora, 2009. 472 p. Disponível em: <<https://books.google.com.br/books?id=d8uCfC46hwsC&lpg=PA2&hl=pt-BR&pg=PA5#v=onepage&q&f=false>>. Acesso em: 24 set. 2017.

SILBERSCHATZ, A; KORTH, H.; SUDARSHAN, S. **Database Systems Concepts**. 4 ed. Boston: McGraw-Hill, 2001.

SMITH, Peter G. **Professional Website Performance**: Optimizing the Front-End and Back-End. Indianapolis, In: John Wiley & Sons, Inc., 2012. 480 p. Disponível em: <[http://pdf.th7.cn/download/files/1312/Professional Website Performance.pdf](http://pdf.th7.cn/download/files/1312/Professional_Website_Performance.pdf)>. Acesso em: 13 nov. 2017.

SOMASUNDARAM, G.; SHRIVASTAVA, Alok; SERVICES, Emc Education. **Armazenamento e Gerenciamento de Informações**: Como armazenar, gerenciar e proteger informações digitais. São Paulo: Bookman Editora, 2009. 472 p.

SOUZA, Terezinha Batista de; CATARINO, Maria Elizabete; SANTOS, Paulo Cesar dos. **Metadados: Catalogando Dados na Internet**. Transinformação-issne 2318-0889, São Paulo, v. 9, n. 2, p.93-105, maio 1997. Disponível em: <<https://seer.sis.puc-campinas.edu.br/seer/index.php/transinfo/article/download/1586/1558>>. Acesso em: 24 set. 2017.

TAYLOR, Allen G. **SQL Para Leigos**. 8. ed. Rio de Janeiro: Alta Books Editora, 2016. 471 p. (Para Leigos). Disponível em: <<https://books.google.com.br/books?id=z1VjDAAAQBAJ>>. Acesso em: 29 nov. 2017.

W3C. **Cascading Style Sheets home page**. 2017. Disponível em: <<http://www.w3.org/Style/CSS/>>. Acesso em: 14 nov. 2017.

W3C. **The HTML Syntax**. 2017. Disponível em: <<https://dev.w3.org/html5/spec-author-view/syntax.html>>. Acesso em: 14 nov. 2017.

APÊNDICES

APÊNDICE A – ARTIGO

Geração de Interfaces de Usuário para Operações CRUD com Base em Metadados

Edinei J. Bauer¹, Fabrício Giordani²

¹Acadêmico do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brasil

²Professor do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brasil

edineibauer@gmail.com, fgjordani@gmail.com

Abstract. *To manage the storage of data through a system, it is necessary to develop User Interfaces (UI) for records insertion, retrieval, deletion and update (CRUD) operations. The development of UI for CRUD operations can account for one-third of the development time of web systems. In order to fully automate the development of UI for CRUD operations, this work proposes the creation of a tool for code generation based on models. To achieve this goal, the tool uses a template engine powered by metadata dictionaries to generate source code, inhibiting the development of algorithms and code refactoring throughout the evolution of the system.*

Resumo. *Para gerenciar o armazenamento de dados através de um sistema, é necessário o desenvolvimento de Interfaces de Usuários (UI, User Interface) para operações de Inserção, Obtenção, Exclusão e Atualização (CRUD - Create, Retrieve, Update and Delete) dos registros. O desenvolvimento de UI para operações CRUD pode representar um terço do tempo de desenvolvimento de sistemas web. Objetivando automatizar completamente o desenvolvimento de UI para operações CRUD, este trabalho propõe a criação de uma ferramenta para geração de códigos baseada em modelos. Para alcançar este objetivo, a ferramenta utiliza um motor de templates alimentada por dicionários de metadados para gerar código-fonte, inibindo o desenvolvimento de algoritmos e a refatoração do código ao longo da evolução do sistema.*

1. Introdução

Automatizar parte do desenvolvimento de sistemas é algo natural (AREND, 2011), que melhora a produtividade, e garante a padronização para o desenvolvimento flexível. Nas últimas 3 décadas, diversos projetos foram desenvolvidos, entretanto, de forma geral, ainda não foi obtido os resultados desejados, os quais se referem principalmente à facilidade de uso, rapidez de desenvolvimento e integração ao ambiente de programação (PUERTA e EISENSTEIN, 1999, tradução nossa).

O uso de dicionário de metadados possibilita o armazenamento das informações das entidades do sistema, ao qual a ferramenta faz uso para gerenciar as tabelas no Sistema Gerenciador de Banco de Dados (SGBD) e para suprir o motor de templates com variáveis de entrada para a geração de modelos de código-fonte. Dentre os diversos tipos de dados presentes nos formulários, e das inúmeras possibilidades de interação com a interface, o

desenvolvimento e padronização de tipos de dados é necessário para cada hipótese. Todos os tipos de dados presentes na ferramenta são declarados em um arquivo, sendo composto por um modelo de template e um dicionário de metadados pré-definido, além de possíveis configurações exclusivas para validação dos dados, manipulação do SGBD, controle da interação do usuário, e estilos personalizados.

Com os tipos de dados definidos, os templates associados e os modelos de Mapeamento Objeto-Relacional (MOR) criadas para cada tipo de dado presente no SGBD. É necessário apenas criar os dicionários de metadados para cada entidade do sistema para que todos os recursos necessários estejam presentes para o uso da ferramenta.

2. Ferramenta para Geração de UI para Operações CRUD

A ferramenta automatiza completamente a geração das interfaces de usuários para operações CRUD, podendo reduzir o tempo de desenvolvimento de um sistema web em até 30% (MRACK, 2007). A fim de abranger a automação completa de todas as telas de cadastros, um conjunto maduro de tipos de dados deve ser desenvolvido para uso nos dicionários de metadados, e eventualmente, devido a derivada necessidade do mercado, novos algoritmos deverão ser acrescentados a ferramenta. O uso de dados relacionais e do armazenamento de arquivos é suportado pela ferramenta, não limitando seu uso por conta das diferentes necessidades na declaração de uma entidade. O reaproveitamento dos tipos de dados desenvolvidos torna o uso da ferramenta mais prática com o passar do tempo.

2.2. Requisitos

A conexão com o SGBD Mysql é a primeira necessidade presente para a utilização da ferramenta. Após, é necessário criar os dicionários de metadados para cada entidade presente no sistema, as tabelas e relações presentes no banco de dados são criadas e gerenciadas pela ferramenta que faz uso das informações presentes no dicionário de metadados para a sua manipulação e definição.

2.3. Funcionamento da Ferramenta

A ferramenta desenvolvida em PHP recebe por parâmetro o dicionário de metadados da entidade ao qual deseja-se gerar a interface de usuário para operações CRUD. A estrutura retornada pela ferramenta não é de fato gerada, todos os códigos são previamente desenvolvidos no formato de templates, e o dicionário de metadados filtra os códigos, moldando e retornando apenas os códigos necessários.

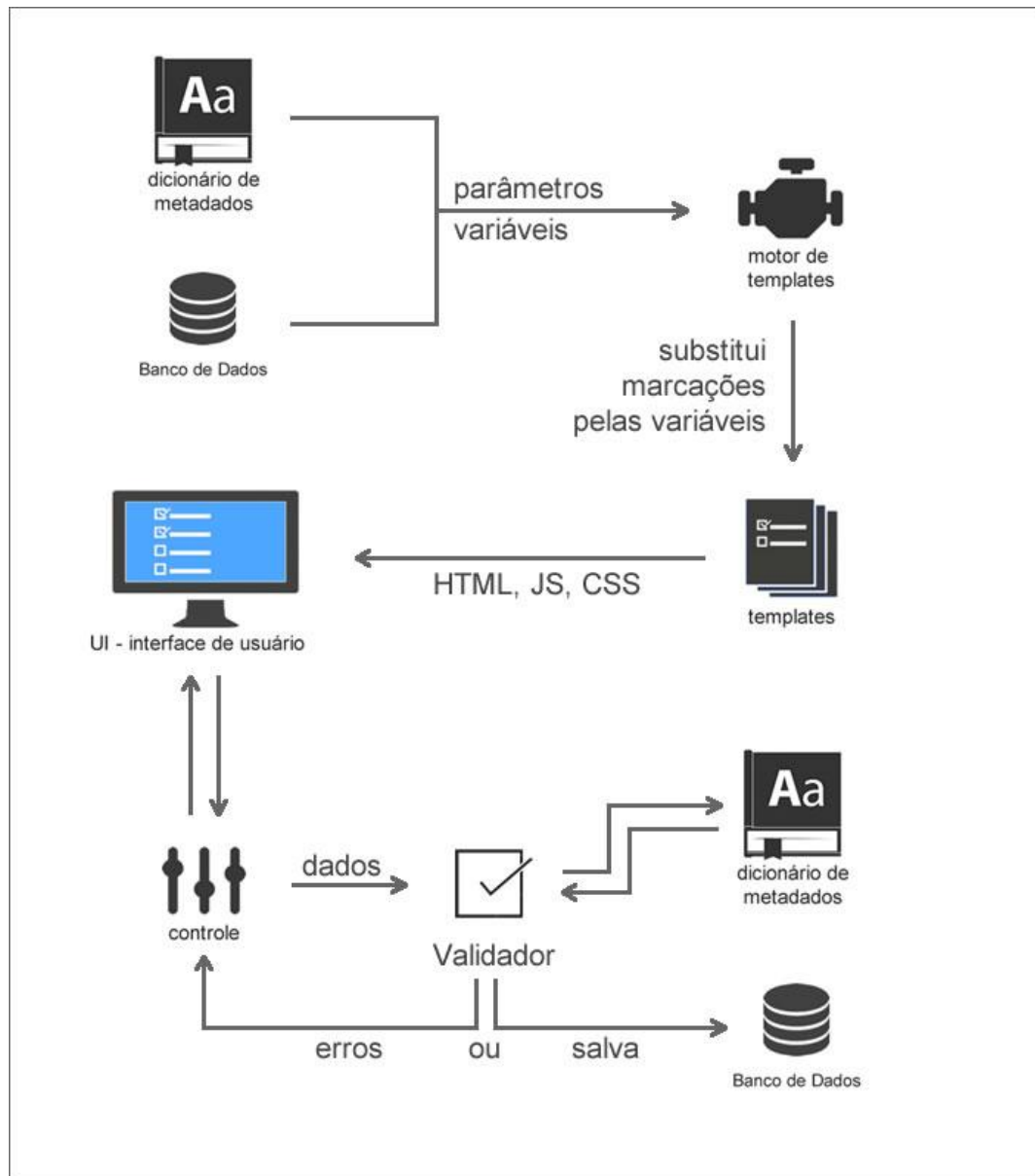


Figura 1. Fluxo da ferramenta para Geração da UI para operações CRUD

2.4. Dicionário de Metadados

Metadados é qualquer dado que tem como objetivo descrever outro dado (AREND, 2011; SOUZA; CATARINO; SANTOS, 1997). Metadados trazem contexto aos dados, permitindo que sejam manipulados dentro de uma aplicação. A quantificação de metadados define o quanto o dado será preciso e amplo, permitindo um maior controle de uso. Dados bem descritos pelo uso de metadados, podem ser utilizados em múltiplos interesses e torna a recuperação e compartilhamento de recursos heterogêneos mais eficientes (ALVES, 2010; SOUZA; CATARINO; SANTOS, 1997).

A principal finalidade dos metadados é documentar e organizar de forma estruturada e relacional os dados, permitindo assim, colaborar com o desenvolvimento ágil, minimizar esforços, duplicações e facilitar a manutenção dos dados (SOUZA; CATARINO; SANTOS, 1997). Para a organização dos metadados, normalmente faz-se uso de um dicionário de metadados.

No sistema, os dicionários de metadados são representados por arquivos no formato *json*, e são mantidos no diretório *entity/cache*, constituídos por listas de pares de chave/valor, onde a primeira lista presente no dicionário representa os campos da entidade, sendo a chave um identificador numérico único do campo, e o valor uma lista de atributos no formato chave/valor. Cada campo representa uma informação a ser armazenada no SGBD, e seus atributos determinam o formato do dado a ser armazenado, além de delimitar e restringir os valores aceitos pelo campo. Os atributos controlam por exemplo, o estilo do campo na interface de usuário, seu nome, tipo de dado, valor padrão, etc.

```

1 {
2   "1": {
3     "format": "title",
4     "type": "varchar",
5     "nome": "Nome",
6     "column": "nome",
7     "size": "127",
8     "key": "title",
9     "unique": "true",
10    "default": "false",
11    "update": "true",
12    "relation": "",
13    "allow": {
14      "regex": "",
15      "validate": ""
16    },
17    "form": {
18      "input": "text",
19      "cols": "12",
20      "colm": "",
21      "coll": "",
22      "class": "",
23      "style": "font-size:1.7em;font-weight:bold"
24    }
25  },
26  "2": {
27    "format": "link",
28    "type": "varchar",
  
```

Figura 2. Modelo do Dicionário de Metadados

2.5 Tipo de Arquivo JSON

JSON é um formato de texto derivado de objetos em JavaScript, para a serialização de dados estruturados entre linguagens de programação definido a partir da terceira geração do ECMAScript Programming Language Standard (ECMA-262) (BRAY, 2014, tradução nossa). JSON foi desenhado dentro de um modelo que fosse mínimo, portátil, textual e um subconjunto de JavaScript, para ser facilmente lido e escrito por seres humanos, e simples de ser interpretado e gerado por máquinas (ECMA INTERNATIONAL, 2013, tradução nossa). Não depende de qualquer linguagem, pois utiliza convenções que são familiares às linguagens C, C++, C#, Java, JavaScript, entre outras.

2.5. Armazenamento dos Dados no SGBD

A configuração das tabelas e de seus campos, assim como suas relações no banco de dados são totalmente gerenciadas pela ferramenta tendo como base as informações presentes no dicionário de metadados. Outras funções do banco de dados Mysql como *views*, *triggers* e

procedures, não são abordadas pela ferramenta, sendo assim, caso tenha a necessidade de seu uso, o mesmo deverá ser configurado manualmente. Qualquer alteração manual realizada nas tabelas do banco de dados poderá resultar no mal funcionamento da ferramenta.

O SGBDR utilizado pela ferramenta é o Mysql 5.7.14, com suporte a dados do tipo *json*, formato cujo qual é adotado pela ferramenta para armazenar determinados tipos de campo. Caso a versão do Mysql não possua suporte a dados do tipo *json*, o mesmo será tratado como um dado *varchar*. Verificações são realizadas no banco de dados quando um dicionário de metadados é utilizado, identificando se a tabela relacionada com o nome do dicionário existe. Caso não exista, a tabela e seus respectivos campos são criados de acordo com os dados presentes no dicionário de metadados. Em campos relacionais, caso a tabela relacional não exista, a ferramenta busca pelo dicionário de metadados desta entidade relacional para adicioná-la primeiro ao banco de dados e posteriormente seguir com a criação da entidade atual. Caso a tabela já exista, uma verificação é realizada para identificar a existência de alterações nos campos da tabela. Alterações nos campos de uma entidade no banco de dados que já possuem dados armazenados, podem gerar erros de alteração ou erros nos dados armazenados, caso as novas configurações dos campos entrem em conflito com os dados já existentes.

2.6. Tipos de Dados

Cada tipo de dado representa um campo na interface de usuário, assim como seu estilo, interação com o usuário e valor armazenado, para cada campo necessário na UI para operações CRUD, um tipo de dado deve ser associado. Os tipos de dados são declarados no arquivo *entity/input_type.json*. Além da definição do tipo de dado, um conjunto predefinido de metadados deve ser associado para aplicação padrão em dados do tipo. Cada campo declarado em um dicionário de metadados deve ser associado a um tipo de dado disponibilizado no arquivo *entity/input_type.json*.

Os tipos de dados trazem um conjunto de definições para serem aplicados a UI, sendo assim, não necessariamente precisam armazenar valores. Eles podem ter o objetivo de armazenar dados brutos como caracteres, números e booleanos, ou dados semânticos, que representam algo por si só, como por exemplo cpf, telefone, email, etc. Além disso, os tipos de dados podem ter por finalidade, estabelecer relação entre dados de diferentes entidades, ou ainda, podem nem armazenar informação, servindo apenas para mostrar informação ao usuário ou interagir com o usuário.

Endereço

Endereço

Avenida Silva Jardim

Rua Dario Pederneiras

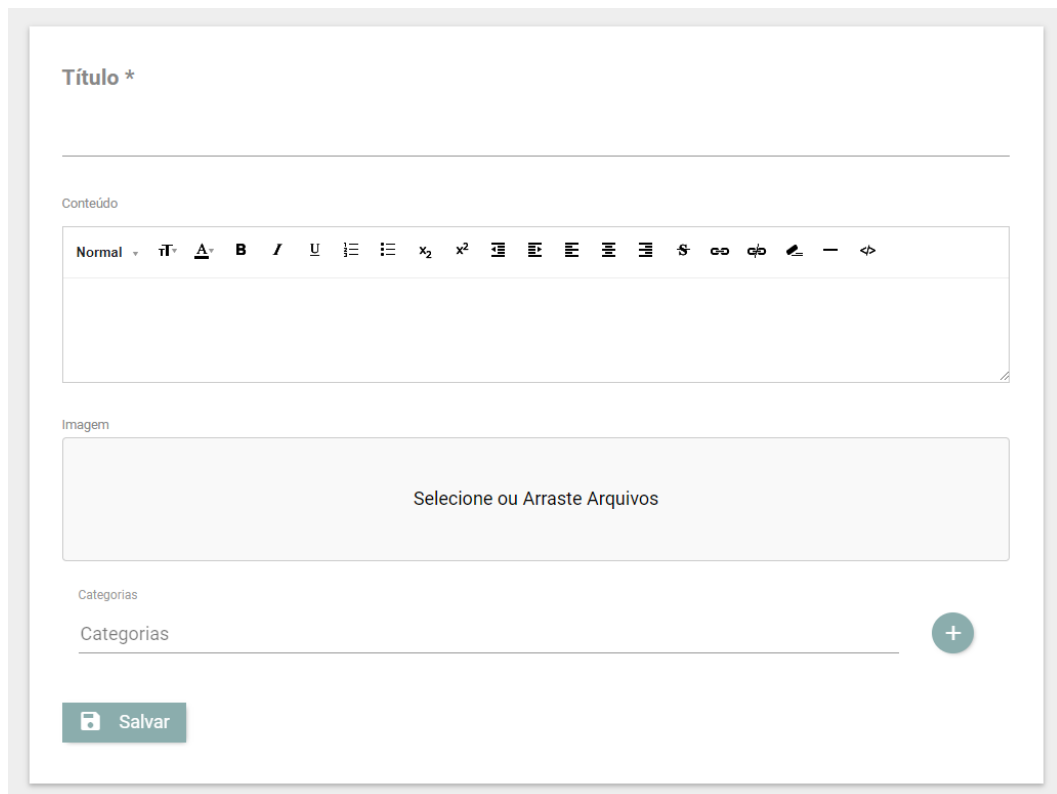
Figura 3. Tipo de dado relacional. Campo traz dados da entidade relacional para associação

2.7. Motor de Templates

Dentro do contexto de aplicações web, um motor de templates é um software que substitui marcações existentes em um arquivo de template por conteúdos dinamicamente carregados pela aplicação (GARCÍA; CASTANEDO; FUENTE, 2007, tradução nossa). O *Smarty* é um motor de templates leve, simples e de fácil implementação, que considera todos os tipos de arquivos com extensão *tpl* como templates. Todos os templates utilizados pela ferramenta estão localizadas na raiz do sistema, no diretório *tpl*.

Os templates surgiram da necessidade de arquivos dinamicamente transformados com o intuito de facilitar o desenvolvimento, melhorar a flexibilidade e reduzir custos de manutenção. São aplicados principalmente em documentos, e-mails, PDF (*Portable Document Format*) e páginas HTML (PARR, 2004, tradução nossa). Um template deve ser escrito da mesma forma que um documento HTML.

Sendo assim, a UI para operações CRUD não é gerada de fato, o motor de templates é o responsável pela mágica que acontece, com base em um conjunto de variáveis de entrada (dicionário de metadados), o motor de templates seleciona e modifica templates presentes na ferramenta para retornar código-fonte personalizado. Para isso, a ferramenta percorre todos os campos do dicionário de metadados, associando um template para cada campo, fazendo a substituição das marcações destes templates pelos atributos presentes no dicionário. Ao final, um formulário com os campos necessários será montado pelo motor de templates, e retornado ao usuário.



The image shows a web form with the following elements:

- Título ***: A text input field.
- Conteúdo**: A rich text editor with a toolbar containing icons for bold, italic, underline, link, unlink, list, and other text formatting options.
- Imagem**: A file upload area with the text "Selecione ou Arraste Arquivos".
- Categorias**: A dropdown menu with a plus sign button to the right.
- Salvar**: A green button with a floppy disk icon and the text "Salvar".

Figura 4 – Exemplo de UI gerada pela ferramenta para operações CRUD

2.8. Controladores

A função dos controladores é detectar alterações e reagir as ações no formulário, buscando informações, suprimindo o formulário, validando dados e submetendo as informações para o SGBD.

Ao salvar as informações do formulário, a ferramenta revisa os dados submetidos fazendo uso do dicionário de metadados para validar os campos, verificando se o valor informado condiz com os atributos do dicionário de metadados. Caso todos os dados sejam válidos e os dados necessários se encontram informados, a ferramenta salva os dados no banco de dados. Caso contrário, o controlador irá informar ao usuário os campos com os erros encontrados.

2.9. UI personalizada

A fim de atender algumas necessidades presentes no desenvolvimento de UI para operações CRUD, configurações extras podem ser aplicadas a ferramenta. Entre elas, é possível configurar o formato do botão de submissão, delimitar os campos que serão exibidos no formulário e disparar uma ação ao salvar o formulário. Além das configurações extras aplicadas na ferramenta, também é possível aplicar configurações extras nos campos relacionais, delimitando os campos que serão necessários ou filtrando os campos a serem selecionados.

```

1  <?php
2
3  echo "<div class='container-900 padding-64'" .
4      "<div class='col card padding-medium color-white'>";
5
6  $form = new Form("noticias");
7  $form->showForm();
8
9  echo "</div></div>";
10

```

Figura 5. Exemplo de chamada da ferramenta passando como parâmetro o dicionário de metadados *noticias*.

3. Resultados Obtidos

A ferramenta automatiza a criação de UI para operações CRUD, gerando em tempo de execução formulários completos para a manipulação e armazenamento de dados. Agilizando e padronizando o desenvolvimento de sistemas web, apresentando características importantes, como a aplicação de regras de usabilidade, a validação automática dos dados, a aplicação de máscaras e de controle de interação, assim como a reponsividade do layout controlada por metadados.

A geração dos campos presentes na UI estão limitados aos tipos de dados presentes no sistema, porém, é possível adicionar novos tipos de dados para as diversas necessidades presentes no mercado, com o reuso dos tipos de dados, é possível reduzir a necessidade de novos tipos de dados com o tempo, diminuindo o tempo necessário para desenvolvimento de UI para operações CRUD conforme a ferramenta é complementada com novos tipos de dados.

4. Discussão

Em relação a outros trabalhos da mesma área desenvolvidos nos últimos anos, como o trabalho sobre Geração Automática e Assistida de Interfaces de Usuário desenvolvida pelo Mrack (2009), e o trabalho sobre *Geração de Operações CRUD a partir de Metadados* apresentado por Arend (2011), cujo qual apresentam uma solução básica a geração de interfaces de usuário para operações CRUD com base nos metadados obtidos do SGBD. Este trabalho se destaca por trazer uma solução completa a geração de interface de usuário para operações CRUD, permitindo gerar interfaces de usuário não somente com dados brutos, mas

também com relações, dados semânticos, informações e armazenamento de arquivos. Além disso, o SGBD é totalmente controlado pela ferramenta, não sendo necessário seu gerenciamento. Com a declaração dos dicionários de metadados em um arquivo *json*, e separado de qualquer tecnologia, o mesmo pode ser acessado por qualquer linguagem, seja ela *front-end* ou *back-end* para uso das informações sobre os campos presentes nas entidades.

5. Conclusão

O trabalho propôs o desenvolvimento de uma ferramenta para a geração automatizada de UI para operações CRUD em sistemas web. Essa geração ocorre com base em dicionários de metadados servindo como variáveis de entrada para um motor de templates.

Com fundamento de reduzir o tempo de desenvolvimento das interfaces de usuários para operações CRUD em sistemas web que representam cerca de 30% do tempo de desenvolvimento (MRACK, 2009). A ferramenta apresenta uma abordagem diferente no desenvolvimento de UI para operações CRUD, a fim de facilitar e melhorar a flexibilidade da criação de sistemas web.

A geração de UI para operações CRUD mostrou-se complexo ao trabalhar com relacionamentos, pois exige configurações adicionais e uma combinação de metadados para atender as derivadas necessidades do mercado. A inserção de novos metadados ou tipos de dados, exige a reconfiguração da ferramenta para que possa funcionar em todas as áreas de atuação, sendo elas, a criação das tabelas no SGBD, a criação do arquivo de informação, o mapeamento objeto-relacional e as validações necessárias.

A ferramenta proposta se limita a gerar interfaces de usuário com os modelos presentes na aplicação, sendo necessário a adição de novos tipos de dados, templates e configurações para atender as demasiadas necessidades do mercado. Porém, uma vez desenvolvido, o código é reaproveitável em futuros sistemas, por isso, o uso deste tipo de ferramenta tende a uma curva decrescente na linha do tempo de desenvolvimento de interfaces de usuários para operações CRUD.

Apesar das dificuldades encontradas no desenvolvimento de dados relacionais e do armazenamento de arquivos, e da limitação da criação de interface de usuário com os modelos de dados presente, a ferramenta gera de forma automática e em tempo de execução, interface de usuário para operações CRUD tendo como base dicionário de metadados. Inibindo o desenvolvimento de código-fonte, assim como o controle sobre os dados armazenados, tornando desnecessário qualquer manipulação no SGBD para seu funcionamento.

Referências

- ALVES, Rachel Cristina Vesú. **Metadados como Elementos do Processo de Catalogação**. 2010. 132 f. Tese (doutorado) - Universidade Estadual Paulista, Faculdade de Filosofia e Ciências, 2010. Disponível em: <<http://hdl.handle.net/11449/103361>>. Acesso em: 16 nov. 2017.
- AREND, Felipe Gabriel. **Geração de Operações CRUD a partir de Metadados**. 2011. 72 f. TCC (Graduação) - Curso de Curso de Ciência da Computação, Centro de Tecnologia, Universidade Federal de Santa Maria, Santa Maria, 2011. Disponível em: <<http://www-app.inf.ufsm.br/bdtg/arquivo.php?id=153&download=1>>. Acesso em: 24 set. 2017.
- BRAY, Tim. **The Javascript Object Notation (JSON) Data Interchange Format**. 2014. Disponível em: <<https://buildbot.tools.ietf.org/pdf/rfc7158.pdf>>. Acesso em: 02 out. 2017.

- ECMA INTERNATIONAL (Suíça). **The JSON Data Interchange Format**. 2013. Disponível em: <<http://www.json.org/json-pt.html>>. Acesso em: 04 out. 2017.
- GARCÍA, F. J.; CASTANEDO, R. I.; FUENTE, A. A. J. **A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications**. 2007. Springer.
- MRACK, Marcelo. Geração Automática e Assistida de Interfaces de Usuário. 2009. 72 f. TCC (Graduação) - Curso de Computação, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009. Disponível em: <<http://hdl.handle.net/10183/37181>>. Acesso em: 09 maio 2017.
- MRACK, M.. **Merlin: Um Novo Horizonte na Criação das Telas de Cadastro**, 8º Fórum Internacional de Software Livre (FISL). Porto Alegre, 2007.
- PARR, T. **Enforcing Strict Model-View Separation in Template Engines**.
Publicado em: WWW '04: Proceedings of the 13th international conference on World Wide Web. ACM, 2004. Disponível em: <<https://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf>>. Acesso em: 20 nov. 2017.
- PUERTA, A.; EISENSTEIN, J.; **Towards a General Computational Framework for Model-Based Interface Development Systems**. Stanford University, Califórnia. 1999.
- SOUZA, Terezinha Batista de; CATARINO, Maria Elizabete; SANTOS, Paulo Cesar dos. **Metadados: Catalogando Dados na Internet**. Transinformação-issne 2318-0889, São Paulo, v. 9, n. 2, p.93-105, maio 1997. Disponível em: <<https://seer.sis.puc-campinas.edu.br/seer/index.php/transinfo/article/download/1586/1558>>. Acesso em: 24 set. 2017.

APÊNDICE B – TIPOS DE DADOS (INPUT_TYPE.JSON)

```

1  {
2  "default": {
3    "format": "text",
4    "type": "varchar",
5    "nome": null,
6    "column": null,
7    "size": false,
8    "key": null,
9    "unique": false,
10   "default": null,
11   "update": true,
12   "relation": null,
13   "allow": {
14     "regex": null,
15     "values": [],
16     "names": [],
17     "validate": null
18   },
19   "form": {
20     "input": "text",
21     "class": null,
22     "style": null,
23     "cols": 12,
24     "colm": null,
25     "coll": null
26   },
27   "select": [],
28   "filter": [],
29   "value": null
30 },
31 "text": {
32   "size": 127
33 },
34 "textarea": {
35   "format": "textarea",
36   "type": "varchar",
37   "nome": "Descrição",
38   "size": 1023,
39   "key": "description",
40   "form": {
41     "input": "textarea"
42   }
43 },
44 "html": {
45   "format": "html",
46   "type": "text",
47   "nome": "Conteúdo",
48   "key": "html",
49   "form": {
50     "input": "html"
51   }
52 },
53 "status": {
54   "format": "status",
55   "type": "tinyint",
56   "nome": "Status",
57   "size": 1,
58   "default": 0,
59   "allow": {
60     "values": [0,1],
61     "names": ["desativado", "ativado"]
62   },
63   "form": {
64     "input": "switch"
65   }
66 },
67 "datetime": {
68   "format": "datetime",
69   "type": "datetime",
70   "nome": "Data",
71   "default": "datetime",
72   "form": {
73     "input": "datetime"
74   }
75 },
76 "title": {
77   "format": "title",
78   "nome": "Título",
79   "size": 127,
80   "key": "title",
81   "unique": true,
82   "default": false,
83   "form": {
84     "style": "font-size:1.7em;font-weight:bold"
85   }
86 },

```

```

87  "link": {
88    "format": "link",
89    "type": "varchar",
90    "nome": "name",
91    "size": 127,
92    "key": "link",
93    "unique": true,
94    "default": false,
95    "form": false
96  },
97  "select": {
98    "format": "select",
99    "type": "tinyint",
100   "default": 1,
101   "allow": {
102     "values": [1,2],
103     "names": ["opção 1", "opção 2"]
104   },
105   "form": {
106     "input": "select"
107   }
108 },
109 "identifiser": {
110   "format": "identifiser",
111   "type": "int",
112   "nome": "id",
113   "key": "identifiser",
114   "unique": true,
115   "default": false,
116   "update": false,
117   "form": {
118     "input": "hidden"
119   }
120 },
121 "extend": {
122   "format": "extend",
123   "type": "int",
124   "key": "extend",
125   "unique": true,
126   "default": false,
127   "update": false,
128   "form": {
129     "input": "extend",
130     "fields": "",
131     "defaults": ""
132   }
133 },
134 "extend_mult": {
135   "format": "extend_mult",
136   "type": "json",
137   "key": "extend_mult",
138   "form": {
139     "input": "extend_mult",
140     "fields": "",
141     "defaults": ""
142   }
143 },
144 "list": {
145   "format": "list",
146   "type": "int",
147   "key": "list",
148   "form": {
149     "input": "list",
150     "fields": "",
151     "defaults": ""
152   }
153 },
154 "selecao": {
155   "format": "selecao",
156   "type": "int",
157   "key": "selecao",
158   "form": {
159     "input": "selecao",
160     "fields": "",
161     "defaults": ""
162   }
163 },
164 "list_mult": {
165   "format": "list_mult",
166   "type": "json",
167   "key": "list_mult",
168   "form": {
169     "input": "list_mult"
170   }
171 },
172 "selecao_mult": {
173   "format": "selecao_mult",
174   "type": "json",
175   "key": "selecao_mult",
176   "form": {
177     "input": "selecao_mult"
178   }
179 },

```

```

180   "source": {
181     "format": "source",
182     "type": "json",
183     "nome": "Imagem",
184     "key": "source",
185     "allow": {
186       "values": [
187         "png",
188         "jpg",
189         "jpeg",
190         "gif",
191         "bmp",
192         "tif",
193         "tiff",
194         "psd",
195         "svg"
196       ],
197       "names": [
198         "png",
199         "jpg",
200         "jpeg",
201         "gif",
202         "bmp",
203         "tif",
204         "tiff",
205         "psd",
206         "svg"
207       ]
208     },
209     "form": {
210       "input": "file"
211     }
212   },
213   "publisher": {
214     "format": "publisher",
215     "type": "int",
216     "key": "publisher",
217     "update": false,
218     "form": false
219   },
220   "email": {
221     "format": "email",
222     "type": "varchar",
223     "nome": "Email",
224     "size": 127,
225     "unique": true,
226     "default": false,
227     "allow": {
228       "validate": "email"
229     },
230     "form": {
231       "input": "email"
232     }
233   },
234   "password": {
235     "format": "password",
236     "type": "varchar",
237     "nome": "Senha",
238     "size": 254,
239     "default": false,
240     "form": {
241       "input": "password"
242     }
243   }
244 }

```


APÊNDICE C – TEMPLATES

DATETIME:

```
<div class="{ $form['class']}" style="{ $form['style']}">
  <label for="{ $ngmodel}">
    { $nome } { ( $default === false ) ? "*" : "" }
  </label>
  <input type="datetime-local" id="{ $ngmodel}" data-model="{ $ngmodel}" data-
format="datetime"
    { ( $value != "" ) ? "value='{ $value}' " : "" }
    { ( $disabled ) ? "disabled='disabled' " : "" }
    { ( $default === false ) ? 'required="required" ' : "" } />
</div>
```

EMAIL:

```
<div class="{ $form['class']}" input-field col s12" style="{ $form['style']}">
  <label class='font-light' for="{ $ngmodel}">
    Email { ( $default === false ) ? "*" : "" }
  </label>
  <input type='email' id="{ $ngmodel}" data-model="{ $ngmodel}" data-format="email"
autocomplete="nope"
    { ( $value != "" ) ? "value='{ $value}' " : "" }
    { ( $size !== false ) ? "maxlength='{ $size}' " : "" }
    { ( $disabled ) ? "disabled='disabled' " : "" }
    { ( $default === false ) ? 'required="required" ' : "" } />
</div>
```

EXTEND:

```
<div class="{ $form['class']}" style="{ $form['style']}" id="parent_{ $column}">
  {foreach $inputs as $input}
    { $input }
  {/foreach}
</div>
```

EXTEND_MULT:

```

<div class="{ $form['class']}" style="{ $form['style']}">
  <div class="row card">
    <div class="col container padding-small right" style="width:160px">
      <button class="btn opacity hover-opacity-off theme-d2 listButton hover-shadow margin-small"
        {($disabled)? "disabled='disabled' " : ""} id="list-{$relation}" data-
entity="{ $relation}">
        <i class="material-icons left padding-right">add</i>
        <span class="left pd-small">Novo</span>
      </button>
      <input type="hidden" data-model="{ $ngmodel}" id="{ $entity}-{$column}" data-
format="extend_mult"
        {($value)? "value='[{foreach item=id key=i from=$value}{if $i >
0},{/if}{$id.id}]{/foreach}]'" : ""} />
    </div>
    <div class="rest container padding-xlarge relative">
      { $nome}
    </div>
    <div class="container listmult-content">
      {if $value}
        {foreach item=data key=i from=$value}
          <div class="listmult-card" style="border-top: solid 2px #EEE;margin-
bottom: 2px!important;"
            rel="{ $data.id}">
            <div class="col padding-small container" style="width:60px">
              <i class="material-icons padding-medium">{$icon}</i>
            </div>
            <div class="rest padding-small relative">
              <div class="right" style="width: 100px; height: 45px">
                <button id="{ $entity}-{$column}-btn"
onclick="editListMult('{ $relation}', '#{ $entity}-{$column}', { $data.id})"
class="btn-floating hover-shadow color-white opacity hover-opacity-off">
                  <i class="material-icons">edit</i>

```

```

        </button>
        <button onclick="removerListMult('#{$entity}-{$column}',
{$data.id})" class="btn-floating color-hover-text-red hover-shadow color-white opacity
hover-opacity-off">
            <i class="material-icons">delete</i>
        </button>
    </div>
    <div class="left container padding-medium listmult-
title">{$data.title}</div>
    </div>
</div>
{/foreach}
{/if}
</div>
</div>
</div>

```

FILE:

```

<div class="{ $form['class'] } form-file" style="{ $form['style'] }">
    <label for="{ $ngmodel }">
        { $nome } { ( $default === false ) ? "*" : "" }
    </label>
    { if isset( $allow['values'] ) }
        <form action="{ $home }request/post" enctype="multipart/form-data" id="form-
{ $entity }-{$column}"
            class="dropzone card">
            <div class="fallback">
                <input name="file" class="hide" type="file" multiple
                    accept="{ foreach item=name key=i from=$allow['values'] } { if $i >
0 }, { /if }. { $name } { /foreach }"/>
            </div>
            <input type="hidden" name="lib" value="form-crud"/>
            <input type="hidden" name="file" value="save/source"/>
            <input type="hidden" name="entity" value="{ $entity }"/>

```

```

        <input type="hidden" name="column" value="{\$column}"/>
    </form>
    <input type="hidden" data-model="{\$ngmodel}" id="{\$entity}-{\$column}" data-
format="file"
        {(\$value)? "value='{\$value|@json_encode}'" : ""}
        {(\$size !== false)? "maxlength='{\$size}'" : ""} />
    {else}
        <h3>Arquivo não aceita Nenhum Valor</h3>
    {/if}
</div>

```

FORM:

```

{if !$reload}
    <input type='hidden' id='fields-{\$entity}' value='{{(\$fields) ? (\$fields|json_encode) :
    ""}' />
{/if}
<div class='form-control row font-large'>
    <div class="row relative form-crud" id='form_{\$entity}' data-entity="{\$entity}">
        <div class="panel">
            <input type='hidden' rel='title' value='{\$relevant}'>
            {foreach \$inputs as \$input}
                {\$input}
            {/foreach}
            <input type="hidden" value="{\$autoSave}" id="autoSave"/>
            <input type="hidden" value="{\$callback}" id="callbackAction"/>
            {if !$autoSave}
                <div class="col padding-16">
                    <button class="btn theme-d2 hover-shadow opacity hover-opacity-off
{\$saveButton.class}" id="saveFormButton">
                        <i class="material-icons left padding-
right">{\$saveButton.icon}</i>{\$saveButton.text}
                    </button>
                </div>
            {/if}

```

```

</div>

{if !$reload}
  <script>
    (function () {
      var $head = document.getElementsByTagName('head')[0];
      if (document.querySelector("script[data-info='form-crud']") === null) {
        var style = document.createElement('link');
        style.rel = "stylesheet";
        style.href = HOME + 'vendor/conn/form-crud/assets/main.min.css?v='
+ VERSION;
        $head.appendChild(style);

        var script = document.createElement('script');
        script.setAttribute("data-info", "form-crud");
        script.src = HOME + 'vendor/conn/form-crud/assets/main.min.js?v=' +
VERSION;
        $head.appendChild(script);
      } else {
        loadForm('#form_{$entity}');
      }
    })();
  </script>
{/if}
</div>
</div>

```

HIDDEN:

```

<input type='hidden' {($value && $value > 0) ? "value='{$value}' " : "" } data-
model="{ $ngmodel}" id="{ $ngmodel}" data-format="hidden" />

```

HTML:

```

<div class="{form['class']}" style="{form['style']}">
  <label for="{ngmodel}">{nome} {(default === false) ? "" : ""}</label>
  <textarea data-model="{ngmodel}" id="{ngmodel}" data-format="textarea"
    {(size !== false)? "maxlength='{size}' " : ""}
    {(default === false)? 'required="required" ' : ""}
    {(disabled)? "disabled='disabled' " : ""}
    style="height: 142px;" class="flow-text padding-small editorHtml hide"
    placeholder="descrição..." rows="10">{value}</textarea>
</div>

```

LIST:

```

<div class="{form['class']}" style="{form['style']}">
  <div class="row padding-8 color-text-grey font-small">{nome} {(default ===
false) ? "" : ""}</div>
  <div class="row">
    <div class="col right" style="width:60px">
      <div class="col btn-floating {(id !== "")? "color-white" : 'theme-d2'} listButton
opacity hover-shadow hover-opacity-off"
        id="list-{column}" data-entity="{relation}"
        {(disabled)? "disabled='disabled' " : ""}
        style="width:41px">
          <i class="material-icons prefix pointer editList">{(id !== "")? "edit" :
'add'}</i>
        </div>
        <input type="hidden" data-model="{ngmodel}" id="{ngmodel}" data-
format="list"
          {(id !== "")? "value='{id}'" : ""} />
        </div>
      <div class="rest relative">
        <input type="text" placeholder="{nome}" autocomplete="off" id="{column}"
          {(title !== "")? "value='{title}'" : ""}
          {(size !== false)? "maxlength='{size}' " : ""}
          {(disabled)? "disabled='disabled' " : ""}

```

```

        {($default === false)? 'required="required" ' : ''}
        data-entity="{ $relation}" data-parent="{ $entity}"
        class="form-list rest"/>
    <div class="col s12 list-complete" rel="one"></div>
</div>
<div class="multFieldsSelect" id="multFieldsSelect-{ $relation}-
{ $column}">{ $mult}</div>
</div>
</div>
<div class="clear"><br></div>

```

LIST_MULT:

```

<div class="{ $form['class']} row card" style="{ $form['style']}">
    <div class="row padding-medium color-text-grey font-small">{ $nome} {($default
=== false) ? "*" : ""}</div>
    <div class="row">
        <div class="col container right" style="width:92px">
            <div class="col btn-floating theme-d2 opacity hover-opacity-off listButton left"
id="list-{ $relation}" data-entity="{ $relation}"
                {($disabled)? "disabled='disabled' " : ''}
                style="width:41px">
                    <i class="material-icons prefix pointer editList">add</i>
                </div>
                <input type="hidden" data-model="{ $ngmodel}" id="{ $entity}-{ $column}" data-
format="list_mult"
                    {($value)? "value='[{foreach item=id key=i from=$value}{if $i >
0},{/if}{ $id.id}{/foreach}]'" : ''} />
                </div>
                <div class="rest container relative">
                    <input type="text" placeholder="{ $nome}" autocomplete="nope"
id="{ $column}"
                        {($size !== false)? "maxlength='{ $size}' " : ''}
                        {($default === false)? 'required="required" ' : ''}
                        {($disabled)? "disabled='disabled' " : ''}

```

```

        data-entity="{relation}" data-parent="{entity}"
        class="form-list rest"/>
    <div class="col s12 list-complete" rel="mult"></div>
</div>
<div class="container listmult-content">
    {if $value}
        {foreach item=data key=i from=$value}
            <div class="listmult-card" style="border-top: solid 2px #EEE;margin-
bottom: 2px!important;" rel="{data.id}">
                <div class="col padding-small container" style="width:60px">
                    <i class="material-icons padding-medium">{icon}</i>
                </div>
                <div class="rest padding-small relative">
                    <div class="right" style="width: 100px; height: 45px">
                        <button id="{entity}-{column}-btn"
                            onclick="editListMult('{relation}', '#{entity}-{column}',
{data.id})" class="btn-floating color-white hover-shadow opacity hover-opacity-off">
                            <i class="material-icons">edit</i>
                        </button>
                        <button onclick="removerListMult("#{entity}-{column}',
{data.id})" class="btn-floating color-white color-hover-text-red hover-shadow opacity
hover-opacity-off">
                            <i class="material-icons">delete</i>
                        </button>
                    </div>
                </div>
                <div class="left container padding-medium listmult-
title">{data.title}</div>
            </div>
        </div>
    {/foreach}
    {/if}
</div>
</div>
</div>

```


PASSWORD:

```

<div class="{form['class']}" style="{form['style']}">
  <label class='font-light' for="{ngmodel}">Nova Senha {(default === false) ? "*" : ""}</label>
  <input type='password' id="{ngmodel}" data-model="{ngmodel}" data-format="password"
    {(size !== false)? "maxlength='{size}' " : ""}
    {(disabled)? "disabled='disabled' " : ""}
    {(default === false)? 'required="required" ' : ""} autocomplete="new-password" />
</div>

```

SELECAO:

```

<div class="{form['class']}" style="{form['style']}">
  <div class="row padding-medium color-text-grey font-small">{nome} {(default === false) ? "*" : ""}</div>
  <div class="row">
    <div class="hide">
      <input type="hidden" data-model="{ngmodel}" id="{ngmodel}" data-format="list" {(id !== "")? "value='{id}'" : ""} /></div>
    <div class="col s12 container relative">
      <input type="text" placeholder="{nome}" autocomplete="off" id="{column}"
        {(title != "")? "value='{title}'" : ""}
        {(size !== false)? "maxlength='{size}' " : ""}
        {(disabled)? "disabled='disabled' " : ""}
        {(default === false)? 'required="required" ' : ""}
        data-entity="{relation}" data-parent="{entity}"
        class="form-list rest"/>
      <div class="col s12 list-complete" rel="one"></div>
    </div>
    <div class="multFieldsSelect" id="multFieldsSelect-{$relation}-{$column}">{mult}</div>
  </div>
</div>

```

SELECAO_MULT:

```

<div class="{ $form['class'] } row card" style="{ $form['style'] }">
  <div class="row padding-medium color-text-grey font-small">{ $nome } {( $default
=== false) ? "*" : ""}</div>
  <div class="row">
    <div class="hide">
      <input type="hidden" data-model="{ $ngmodel }" id="{ $entity }-{ $column }" data-
format="selecao_mult"
        {( $value) ? "value='[{foreach item=id key=i from=$value}{if $i >
0},{/if}{ $id.id }{/foreach}]'" : '' } />
    </div>
    <div class="col s12 container relative">
      <input type="text" placeholder="{ $nome }" autocomplete="nope"
id="{ $column }"
        {( $size !== false) ? "maxlength='{ $size }' " : '' }
        {( $default === false) ? 'required="required" ' : '' }
        {( $disabled) ? "disabled='disabled' " : '' }
        data-entity="{ $relation }" data-parent="{ $entity }"
        class="form-list rest"/>
      <div class="col s12 list-complete" rel="mult"></div>
    </div>

    <div class="container listmult-content">
      {if $value}
        {foreach item=data key=i from=$value}
          <div class="listmult-card" style="border-top: solid 2px #EEE;margin-
bottom: 2px!important;" rel="{ $data.id }">
            <div class="col padding-small container" style="width:60px">
              <i class="material-icons padding-medium">{ $icon}</i>
            </div>
            <div class="rest padding-small relative">
              <div class="right" style="width: 45px; height: 45px">
                <button onclick="removerListMult('#{ $entity }-{ $column }',
{ $data.id })"

```

```

        class="btn-floating color-white color-hover-text-red hover-
shadow opacity hover-opacity-off"><i
        class="material-icons">delete</i></button>
    </div>
    <div class="left container padding-medium listmult-
title">{$data.title}</div>
    </div>
</div>
    {/foreach}
    {/if}
</div>
    <div class="tpl_list_mult hide listmult-card" style="border-top: solid 2px
#EEE;margin-bottom: 2px!important;" rel="__ $0__">
    <div class="col padding-small container" style="width:60px">
        <i class="material-icons padding-medium">{$icon}</i>
    </div>
    <div class="rest padding-small relative">
        <div class="right" style="width: 45px; height: 45px">
            <button onclick="removeListMult('#{$entity}-{$column}', __ $0__)"
                class="btn-floating color-white color-hover-text-red hover-shadow
opacity hover-opacity-off"><i
                class="material-icons">delete</i></button>
        </div>
        <div class="left container padding-medium listmult-title">__ $1__</div>
    </div>
    </div>
</div>
</div>

```

SELECT:

```

<div class="{ $form['class']}" style="{ $form['style']}">
  <label for="{ $ngmodel}">{$nome} {($default === false) ? "*" : ""}</label>
  <select data-model="{ $ngmodel}" id="{ $ngmodel}" data-format="select"
    {($default === false)? 'required="required" ' : ''}
    {($disabled)? "disabled='disabled' " : ''}>
    <option value="0" {(! $value || ($value && $value !== "")) ? "selected='selected' "
: ""}disabled="disabled">
      seleccione
    </option>
    {foreach key=key item=item from=$allow['values']}
      <option {($value && $value === $item) ? "selected='selected' " :
""}value="{ $item}">{$allow['names'][$key]}</option>
    {/foreach}
  </select>
</div>

```

SWITCH:

```

<div class="{ $form['class']}" style="{ $form['style']}">
  <label for="{ $ngmodel}" class="row">{$nome} {($default === false) ? "*" :
""}</label>
  <label class="switch">
    <input type="checkbox" data-model="{ $ngmodel}" id="{ $ngmodel}" data-
format="switch"
      {($value !== false && $value === 1) ? "checked='checked' " : "" }
      {($size !== false)? "maxlength='{$size}' " : ''}
      {($disabled)? "disabled='disabled' " : ''}
      {($default === false)? 'required="required" ' : ''}
      class="switchCheck"/>
    <div class="slider"></div>
  </label>
</div>

```

TEXT:

```

<div class="{form['class']}" style="{form['style']}">
  <label for="{ngmodel}">{nome} {(default === false) ? "*" : ""}</label>
  <input type="text" data-model="{ngmodel}" id="{ngmodel}" data-format="text"
autocomplete="off"
    {(value != "")? "value='{value}'" : ""}
    {(size !== false)? "maxlength='{size}'" : ""}
    {(disabled)? "disabled='disabled'" : ""}
    {(default === false)? 'required="required"' : ""} />
</div>

```

TEXTAREA:

```

<div class="{form['class']}" style="{form['style']}">
  <label for="{ngmodel}">{nome} {(default === false) ? "*" : ""}</label>
  <textarea data-model="{ngmodel}" id="{ngmodel}" data-format="textarea"
    {(size !== false)? "maxlength='{size}'" : ""}
    {(default === false)? 'required="required"' : ""}
    {(disabled)? "disabled='disabled'" : ""}
    style="height: 142px;" class="flow-text padding-small"
rows="10">{value}</textarea>
</div>

```

APÊNDICE D – DICIONÁRIO DE METADADOS

usuarios:

```

1  {
2    "1": {
3      "format": "title",
4      "type": "varchar",
5      "nome": "Nome de Usuario",
6      "column": "nome_de_usuario",
7      "size": "127",
8      "key": "title",
9      "unique": "true",
10     "default": "false",
11     "update": "true",
12     "relation": "",
13     "allow": {
14       "regex": "",
15       "validate": ""
16     },
17     "form": {
18       "input": "text",
19       "cols": "12",
20       "colm": "",
21       "coll": "",
22       "class": "",
23       "style": ""
24     },
25     "value": "",
26     "indice": "1"
27   },
28   "2": {
29     "format": "link",
30     "type": "varchar",
31     "nome": "Nome Usuario",
32     "column": "nome_usuario",
33     "size": "127",
34     "key": "link",
35     "unique": "true",
36     "default": "false",
37     "update": "true",
38     "relation": "",
39     "allow": {
40       "regex": "",
41       "validate": ""
42     },
43     "form": "false",
44     "value": "",
45     "input": "",
46     "indice": "2"
47   },
48   "3": {
49     "format": "source",
50     "type": "json",
51     "nome": "Imagem",
52     "column": "imagem",
53     "size": "false",
54     "key": "source",
55     "unique": "false",
56     "default": "",
57     "update": "true",
58     "relation": "",
59     "allow": {
60       "regex": "",
61       "values": [
62         "png",
63         "jpg",
64         "jpeg",
65         "gif",
66         "bmp",
67         "tif",
68         "tiff",
69         "psd",
70         "svg"
71       ],
72       "names": [
73         "png",
74         "jpg",
75         "jpeg",
76         "gif",
77         "bmp",
78         "tif",
79         "tiff",
80         "psd",
81         "svg"
82       ],
83       "validate": ""
84     },
85     "form": "false",
86     "value": "",
87     "input": "file",
88     "indice": "10"
89   },

```

```

90  "4": {
91    "format": "email",
92    "type": "varchar",
93    "nome": "Email",
94    "column": "email",
95    "size": "127",
96    "key": "",
97    "unique": "true",
98    "default": "false",
99    "update": "true",
100   "relation": "",
101   "allow": {
102     "regex": "",
103     "validate": "email"
104   },
105   "form": {
106     "input": "email",
107     "cols": "12",
108     "colm": "8",
109     "coll": "",
110     "class": "padding-tiny",
111     "style": ""
112   },
113   "value": "",
114   "indice": "5"
115 },
116 "5": {
117   "format": "password",
118   "type": "varchar",
119   "nome": "Password",
120   "column": "password",
121   "size": "254",
122   "key": "",
123   "unique": "false",
124   "default": "",
125   "update": "true",
126   "relation": "",
127   "allow": {
128     "regex": "\\/^.{4,10}$\\\/i",
129     "validate": ""
130   },
131   "form": {
132     "input": "password",
133     "cols": "12",
134     "colm": "4",
135     "coll": "",
136     "class": "padding-tiny",
137     "style": ""
138   },
139   "value": "",
140   "indice": "7"
141 },
142 "6": {
143   "format": "datetime",
144   "type": "datetime",
145   "nome": "Data",
146   "column": "data",
147   "size": "false",
148   "key": "",
149   "unique": "false",
150   "default": "datetime",
151   "update": "true",
152   "relation": "",
153   "allow": {
154     "regex": "",
155     "validate": ""
156   },
157   "form": "false",
158   "value": "",
159   "input": "text",
160   "indice": "18"
161 },
162 "7": {
163   "format": "status",
164   "type": "tinyint",
165   "nome": "Status",
166   "column": "status",
167   "size": "1",
168   "key": "",
169   "unique": "false",
170   "default": "1",
171   "update": "true",
172   "relation": "",
173   "allow": {
174     "regex": "",
175     "values": [
176       "0",
177       "1"
178     ],
179     "names": [
180       "desativado",
181       "ativado"
182     ],
183     "validate": ""
184   },
185   "form": {
186     "input": "switch",
187     "cols": "12",
188     "colm": "3",
189     "coll": "",
190     "class": "padding-tiny",
191     "style": ""
192   },
193   "value": "",
194   "indice": "19"
195 },

```

```

196 "8": {
197   "format": "select",
198   "type": "tinyint",
199   "nome": "Nível",
200   "column": "nivel",
201   "size": "false",
202   "key": "",
203   "unique": "false",
204   "default": "2",
205   "update": "true",
206   "relation": "",
207   "allow": {
208     "regex": "",
209     "values": [
210       "2",
211       "1"
212     ],
213     "names": [
214       "Administrador",
215       "Desenvolvedor",
216     ],
217     "validate": ""
218   },
219   "form": {
220     "input": "select",
221     "cols": "12",
222     "colm": "9",
223     "coll": "",
224     "class": "padding-tiny",
225     "style": ""
226   },
227   "select": "",
228   "filter": "",
229   "value": "",
230   "indice": "20"
231 },

```

```

232 "9": {
233   "format": "text",
234   "type": "varchar",
235   "nome": "token",
236   "column": "token",
237   "size": "254",
238   "key": "",
239   "unique": "false",
240   "default": "",
241   "update": "true",
242   "relation": "",
243   "allow": {
244     "regex": "",
245     "validate": ""
246   },
247   "form": "false",
248   "value": "",
249   "input": "text",
250   "indice": "22"
251 },
252 "0": {
253   "format": "none",
254   "type": "int",
255   "nome": "id",
256   "column": "id",
257   "size": "",
258   "key": "identifíer",
259   "unique": "true",
260   "default": "false",
261   "update": "false",
262   "relation": "",
263   "allow": {
264     "regex": "",
265     "values": "",
266     "names": "",
267     "validate": ""
268   },
269   "form": {
270     "input": "hidden",
271     "cols": "12",
272     "colm": "",
273     "coll": "",
274     "class": "",
275     "style": ""
276   },
277   "select": [],
278   "filter": []
279 }
280 }

```


noticias:

```

1  {
2    "1": {
3      "format": "title",
4      "type": "varchar",
5      "nome": "Titulo",
6      "column": "titulo",
7      "size": "127",
8      "key": "title",
9      "unique": "true",
10     "default": "false",
11     "update": "true",
12     "relation": "",
13     "allow": {
14       "regex": "",
15       "values": "",
16       "names": "",
17       "validate": ""
18     },
19     "form": {
20       "input": "text",
21       "cols": "12",
22       "colm": "",
23       "coll": "",
24       "class": "",
25       "style": ""
26     },
27     "select": "",
28     "filter": "",
29     "value": "",
30     "indice": "1"
31   },
32   "2": {
33     "format": "link",
34     "type": "varchar",
35     "nome": "name",
36     "column": "name",
37     "size": "127",
38     "key": "link",
39     "unique": "true",
40     "default": "false",
41     "update": "true",
42     "relation": "",
43     "allow": {
44       "regex": "",
45       "values": "",
46       "names": "",
47       "validate": ""
48     },
49     "form": "false",
50     "select": "",
51     "filter": "",
52     "value": "",
53     "input": "text",
54     "indice": "2"
55   },
56   "3": {
57     "format": "html",
58     "type": "text",
59     "nome": "Conteudo",
60     "column": "conteudo",
61     "size": "false",
62     "key": "html",
63     "unique": "false",
64     "default": "",
65     "update": "true",
66     "relation": "",
67     "allow": {
68       "regex": "",
69       "values": "",
70       "names": "",
71       "validate": ""
72     },
73     "form": {
74       "input": "html",
75       "cols": "12",
76       "colm": "",
77       "coll": "",
78       "class": "",
79       "style": ""
80     },
81     "select": "",
82     "filter": "",
83     "value": "",
84     "indice": "3"
85   },
86   "4": {
87     "format": "source",
88     "type": "json",
89     "nome": "Imagem",
90     "column": "imagem",
91     "size": "false",
92     "key": "source",
93     "unique": "false",
94     "default": "",
95     "update": "true",
96     "relation": "",
97     "allow": {
98       "regex": "",
99       "values": ["png", "jpg", "jpeg"],
100      "names": ["png", "jpg", "jpeg"],
101      "validate": ""
102    },
103    "form": {"input": "file", "cols": "12"},
104    "select": "",
105    "filter": "",
106    "value": "",
107    "indice": "4"
108  },

```

```

109 "5": {
110   "format": "datetime",
111   "type": "datetime",
112   "nome": "Data",
113   "column": "data",
114   "size": "false",
115   "key": "",
116   "unique": "false",
117   "default": "datetime",
118   "update": "true",
119   "relation": "",
120   "allow": {
121     "regex": "",
122     "values": "",
123     "names": "",
124     "validate": ""
125   },
126   "form": "false",
127   "select": "",
128   "filter": "",
129   "value": "",
130   "input": "datetime",
131   "indice": "5"
132 },
133 "6": {
134   "format": "publisher",
135   "type": "int",
136   "nome": "autor",
137   "column": "autor",
138   "size": "false",
139   "key": "publisher",
140   "unique": "false",
141   "default": "",
142   "update": "false",
143   "relation": "",
144   "allow": {
145     "regex": "",
146     "values": "",
147     "names": "",
148     "validate": ""
149   },
150   "form": "false",
151   "select": "",
152   "filter": "",
153   "value": "",
154   "input": "text",
155   "indice": "6"
156 },
157 "7": {
158   "format": "list_mult",
159   "type": "json",
160   "nome": "Categorias",
161   "column": "categorias",
162   "size": "false",
163   "key": "list_mult",
164   "unique": "false",
165   "default": "",
166   "update": "true",
167   "relation": "categorias",
168   "allow": {
169     "regex": ""
170   },
171   "form": {
172     "input": "list_mult",
173     "cols": "12",
174     "fields": ["1", "3"],
175     "defaults": {
176       "1": "",
177       "2": "",
178       "3": "",
179       "4": ""
180     }
181   },
182   "select": "",
183   "filter": "",
184   "value": "",
185   "indice": "7"
186 },
187 "0": {
188   "format": "none",
189   "type": "int",
190   "nome": "id",
191   "column": "id",
192   "size": "",
193   "key": "identificador",
194   "unique": "true",
195   "default": "false",
196   "update": "false",
197   "relation": "",
198   "allow": {
199     "regex": ""
200   },
201   "form": {
202     "input": "hidden",
203     "cols": "12",
204   },
205   "select": [],
206   "filter": []
207 }
208 }

```

categorias:

```

1  {
2    "1": {
3      "format": "title",
4      "type": "varchar",
5      "nome": "Nome",
6      "column": "nome",
7      "size": "127",
8      "key": "title",
9      "unique": "true",
10     "default": "false",
11     "update": "true",
12     "relation": "",
13     "allow": {
14       "regex": ""
15     },
16     "form": {
17       "input": "text",
18       "cols": "12",
19     },
20     "select": "",
21     "filter": "",
22     "value": "",
23     "indice": "1"
24   },
25   "2": {
26     "format": "link",
27     "type": "varchar",
28     "nome": "link",
29     "column": "link",
30     "size": "127",
31     "key": "link",
32     "unique": "true",
33     "default": "false",
34     "update": "true",
35     "relation": "",
36     "allow": {
37       "regex": ""
38     },
39     "form": "false",
40     "select": "",
41     "filter": "",
42     "value": "",
43     "input": "text",
44     "indice": "2"
45   },
46   "3": {
47     "format": "textarea",
48     "type": "varchar",
49     "nome": "Descricao",
50     "column": "descricao",
51     "size": "1023",
52     "key": "description",
53     "unique": "false",
54     "default": "",
55     "update": "true",
56     "relation": "",
57     "allow": { "regex": "" },
58     "form": { "input": "textarea", "cols": "12" },
59     "select": "",
60     "filter": "",
61     "value": "",
62     "indice": "3"
63   },
64   "4": {
65     "format": "datetime",
66     "type": "datetime",
67     "nome": "Data",
68     "column": "data",
69     "size": "false",
70     "key": "",
71     "unique": "false",
72     "default": "datetime",
73     "update": "true",
74     "relation": "",
75     "allow": { "regex": "" },
76     "form": "false",
77     "select": "",
78     "filter": "",
79     "value": "",
80     "input": "datetime",
81     "indice": "4"
82   },
83   "0": {
84     "format": "none",
85     "type": "int",
86     "nome": "id",
87     "column": "id",
88     "size": "",
89     "key": "identififier",
90     "unique": "true",
91     "default": "false",
92     "update": "false",
93     "relation": "",
94     "allow": { "regex": "" },
95     "form": { "input": "hidden", "cols": "12" },
96     "select": [],
97     "filter": []
98   }
99 }

```