FERRAMENTA GERADORA DE APLICAÇÕES WEB COM JHIPSTER V2: PERSONALIZAÇÃO DE FORMULÁRIOS E INTEGRIDADE DE DADOS

Cleber Tomazoni, Mauro Marcelo Mattos – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

ctomazoni@furb.br, mattos@furb.br

Resumo: *O processo acelerado de transformação digital da sociedade tem gerado uma forte demanda pela produção rápida de soluções em software. Uma das abordagens utilizadas para acelerar o processo de desenvolvimento web utiliza a técnica de scaffolding que consiste na geração da estrutura de código-fonte inicial a partir de metamodelos. O presente trabalho descreve os incrementos que foram realizados sobre a versão inicial de uma ferramenta desenvolvida para acelerar novos projetos, chamada DB2JHipster, a qual é baseada no framework JHipster, gerando o código-fonte de uma aplicação web a partir da leitura de metadados de um banco de dados informado pelo usuário. Além disso, a nova versão permite a migração dos dados da base de origem, facilitando o processo de geração de protótipos rápidos. São apresentados a especificação e o resultado dos testes realizados o que demonstra a efetividade da solução construída.*

Palavras-chave*: JHipster. DB2JHipster. Scaffolding. Metamodelos. Framework.*

1 INTRODUÇÃO

A adoção de formas novas e inovadoras de fazer negócios em organizações com base nos avanços tecnológicos tem sido referida como transformação digital (RED HAT, 2017, p. 1) e diz respeito a todas as ações relacionadas a melhoria em processos como de gestão, de produção, de relacionamento com os clientes e entre fornecedores possibilitando ganhos competitivos. Este movimento gera pressão sobre à área de tecnologia da informação, tendo em vista que o surgimento de novas oportunidades de interação com o cliente demanda um esforço de desenvolvimento de software não necessariamente compatível com o conjunto de habilidades que a equipe técnica dispõe ou que a infraestrutura atual permite (MACHADO, 2019; MUNDIM; SIESTRUP, 2019, p. 45; NASCIMENTO, 2020, p. 1).

Analisando-se este cenário sob o ponto de vista da área de engenharia de software, observa-se que os sistemas computacionais oferecem soluções de problemas normalmente parecidos e que são recorrentes, tornando a reutilização do código algo bastante útil (ABREU, 2016, p. 36). É neste contexto que surgem os chamados *frameworks* que se caracterizam como uma arquitetura de software organizada de tal forma a permitir máxima reutilização possível, sendo representada como um conjunto de classes abstratas e concretas, bastante genéricas para possibilitar um grande potencial de especialização (MATTSSON, 1996, p. 51). As principais características dos *frameworks* são: modularidade, reutilização, extensibilidade e eventualmente inversão de controle (quando assumem o controle da execução invocando métodos da aplicação quando necessário) (VOLPON, 2011, p. 46).

Neste contexto surge a ferramenta JHipster (DUBOIS; SASIDHARAN; GRIMAUD, 2020), um gerador de código aberto usado industrialmente para o desenvolvimento de aplicativos da web, o qual usa a técnica de *scaffolding* para produzir a estrutura de diretórios da aplicação. A partir de uma configuração especificada pelo usuário, o JHipster gera uma pilha tecnológica completa constituída de código Java e Spring Boot (no lado do servidor) e Angular e Bootstrap (no lado cliente). O gerador suporta várias tecnologias, desde o banco de dados usado (por exemplo, MySQL ou MongoDB), o mecanismo de autenticação (por exemplo, HyperText Transfer Protocol (HTTP) Session ou Oauth2), o suporte ao *logon* social (via contas de redes sociais existentes) e o uso de microsserviços (HALIN *et al.*, 2018). No entanto, o conjunto de possibilidades de diferentes configurações do JHipster o torna ao mesmo tempo uma ferramenta que traz um ganho de produtividade, mas pode acarretar uma série de efeitos colaterais como demonstrado na seção 2.1 deste trabalho.

Na perspectiva de facilitar o processo de geração correta do arquivo de configurações do JHipster, Aguiar (2019) desenvolveu como trabalho de conclusão de curso um protótipo capaz de gerar este arquivo a partir das definições em Linguagem de Definição de Dados (LDD) das tabelas de um banco de dados. Diante do exposto, o presente trabalho tem como objetivo geral incrementar o projeto de Aguiar (2019) tornando-o uma ferramenta mais adequada para utilização em ambiente de produção. Como objetivos específicos, o projeto deve (a) aprimorar os aspectos de conexão com as bases de dados selecionadas e (b) introduzir uma camada de validação dos relacionamentos identificados no modelo de dados.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem o objetivo de explorar os assuntos mais relevantes para a realização deste trabalho. Inicialmente é discorrido sobre a complexidade de sistemas configuráveis na seção 2.1, posteriormente a seção 2.2 apresenta a tecnologia JHipster, utilizada para a geração de código-fonte. O protótipo desenvolvido por Aguiar (2019) é apresentado na seção 2.3. A seção 2.4 traz os trabalhos correlatos.

2.1 A COMPLEXIDADE DE SISTEMAS CONFIGURÁVEIS

Um sistema configurável é um software que possui um conjunto base de funcionalidades e um conjunto variável de facilidades que podem ser definidas por um conjunto de opções de configuração (ou preferencias) em que as mudanças no valor de uma preferência alteram o comportamento do software de alguma forma (JIN *et al.*, 2014, p. 216). Se por um lado, a granularidade de possibilidades de configurações traz um benefício para o desenvolvedor (ou usuário final, dependendo do escopo), também introduz muitos desafios durante a etapa de testes e/ou depuração de um sistema, uma vez que a complexidade e a inter-relação das variáveis complicam o processo de localizar e/ou reproduzir uma falha. Bettenburg *et al.* (2008) reportam que em sistemas configuráveis há um desalinhamento entre o que os desenvolvedores necessitam para reproduzir e corrigir um erro e o que é fornecido pelos usuários.

Dada a complexidade dos sistemas de software atuais, determinar o espaço de configuração pode não ser uma tarefa trivial. Em seu trabalho Jin *et al.* (2014) reportam que o Firefox, o navegador de código aberto tem mais de 1.900 opções de configuração disponíveis para um usuário. O espaço de possíveis configurações exclusivas cresce exponencialmente com o número de opções (ou preferências) de configuração, portanto, podemos avaliar apenas uma amostra representativa de todas as configurações possíveis. E Jin *et al.* (2014) complementam que esta complexidade está associada ao fato de que, em muitos casos, as configurações podem ser modificadas e visualizadas em vários locais. Estas configurações são encontradas em estruturas dinâmicas e estáticas e é possível que as estruturas estáticas estejam fora de sincronização com as dinâmicas no momento da falha (JIN *et al.*, 2014, p. 216).

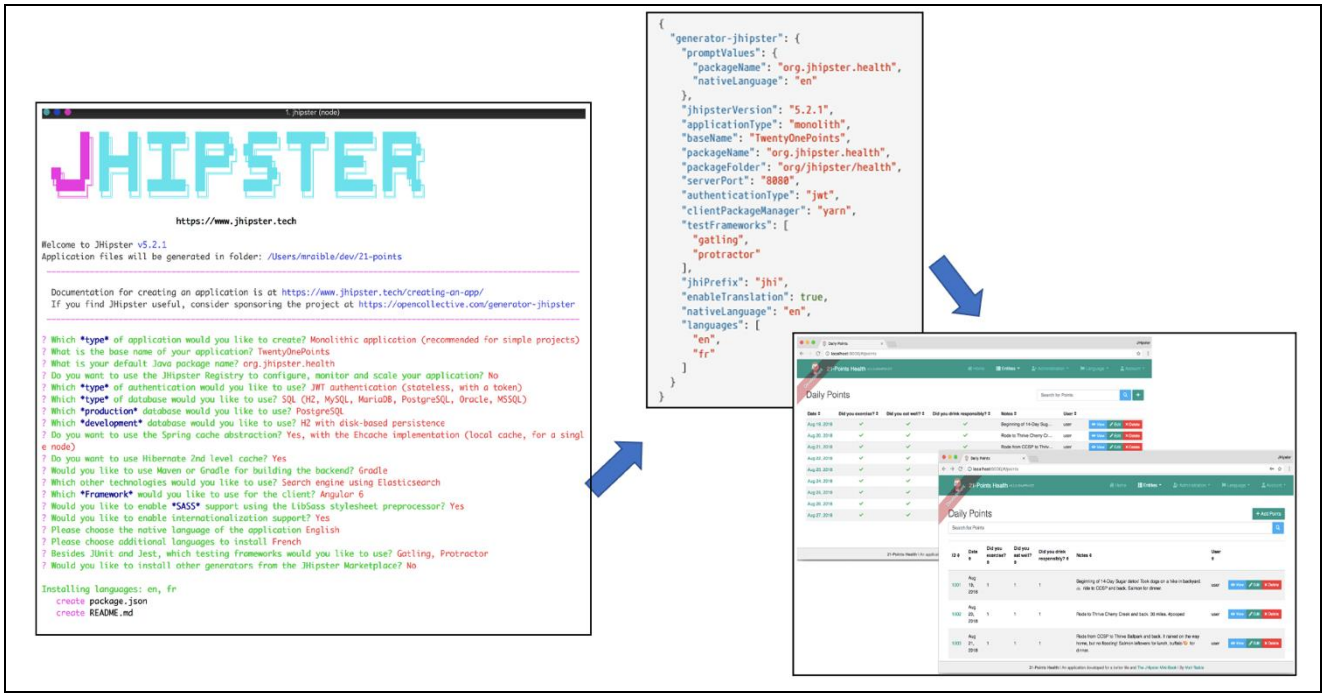
Conforme Halin *et al.* (2018), o JHipster é um sistema configurável complexo com as seguintes características: i) uma variedade de linguagens (JavaScript, Cascading Style Sheets (CSS), Structured Query Language (SQL) etc.) e tecnologias avançadas (Maven, Docker, etc.) são combinadas para gerar variantes; ii) existem 48 opções de configuração e um configurador que guia o usuário em diferentes perguntas (embora nem todas as combinações de opções sejam possíveis além de existirem 15 restrições entre as opções); iii) a variabilidade está espalhada entre vários tipos de artefatos (pom.xml, classes Java, arquivos Docker, etc.) e várias opções normalmente contribuem para a ativação ou desativação de partes do código, o que é comumente observado em software configurável (JIN *et al.*, 2014). Assim, ao mesmo tempo em que se caracteriza como uma ferramenta de produtividade, um estudo de Halin *et al.* (2018) produziu mais de 26 mil possibilidades de configurações diferentes e constatou que aproximadamente 36% delas resultaram em combinações não funcionais dos recursos o que torna este um campo aberto de investigação.

2.2 JHIPSTER

O JHipster é uma plataforma de desenvolvimento concebida para facilitar o processo de desenvolvimento de aplicações web e arquiteturas baseadas em microsserviços. O *frontend* das aplicações pode ser desenvolvido com as tecnologias Angular, React e Vue, enquanto o *backend* suporta Spring Boot, .NET e Node.js (JHIPSTER, 2020, p. 1). Para otimizar o desenvolvimento, o JHipster disponibiliza três tipos de perfis para executar a aplicação: i) desenvolvimento: usado para a criação do projeto; ii) produção: para colocar a aplicação em produção e, iii) *fast*: para iniciar a aplicação mais rapidamente, desabilitando alguns recursos (PRATES JUNIOR, 2016, p. 1). Um dos objetivos do JHipster é disponibilizar uma interface de usuário elegante baseada em Bootstrap por meio de um fluxo de trabalho que viabiliza rapidamente a criação de aplicações e uma infraestrutura que possa ser implantada facilmente na nuvem (JHIPSTER, 2020, p. 1; RAIBLE, 2019, p. 1).

O JHipster constrói um arquivo de configuração a partir de uma aplicação de linha de comando (Figura 1), no qual o desenvolvedor escolhe dentre as opções suportadas, quais componentes devem ser utilizados para gerar para o aplicativo. Neste ponto, o modelo de domínio ainda não é expresso nem criado, somente o arquivo de configuração está gerado em uma linguagem denominada JHipster Domain Language (JDL) (LIMA, 2016). Após montar a estrutura da aplicação, o JHipster recebe as instruções para o modelo de domínio pretendido para a aplicação. Conforme citado por Aguiar (2019, p. 17), o JHipster gera para cada entidade criada, os seguintes componentes: tabela de banco de dados com o respectivo registro de mudanças, classe de entidade Java Persistence API (JPA), interface Spring JPA Repository, classe Spring Model-View-Controller (MVC) REpresentational State Transfer (REST) Controller e a *view* em Angular, contendo rotas, controlador, serviço e a página HyperText Markup Language (HTML) relacionada.

Figura 1 - Configuração de uma aplicação JHipster e exemplo de aplicação gerada



Fonte: adaptado de Raible (2019).

A linguagem JDL possui elementos para a especificação de entidades e relacionamentos conforme a sintaxe apresentada no Quadro 1 (JHIPSTER, 2019, p. 1) em que:

a) `<entity name>`: é o nome da entidade;
b) `<field name>`: é o nome de um campo da entidade;
c) `<field type>`: o tipo Jhipster suportado do campo da entidade;
d) `<validation>`: validações para o campo (não será abordado nesse trabalho)
e) `<entity javadoc>` e `<field javadoc>`: são opções de documentação para a entidade e os campos.

Quadro 1 - Sintaxe de declaração de uma entidade

```
[<entity javadoc>]
entity <entity name> [(<table name>)] {
  [<field javadoc>]
  <field name> <field type> [<validation>*]
}
```

Fonte: JHipster (2019, p. 1).

Conforme Silva Neto e Vilar (2019, p. 2), embora *frameworks* como Ruby on Rails, Django e JHipster facilitarem o desenvolvimento de aplicações web, eles não geram automaticamente alguns aspectos de segurança fazendo com que o programador precise intervir no código fonte gerado para implementar os requisitos de segurança desejados. Um outro aspecto importante é que a especificação do modelo demanda domínio da estrutura da linguagem JDL. A próxima seção apresenta o protótipo desenvolvido por Aguiar (2019), no qual este aspecto é superado permitindo que o modelo da aplicação seja gerado automaticamente a partir de um modelo de banco de dados existente.

2.3 DB2JHIPSTER

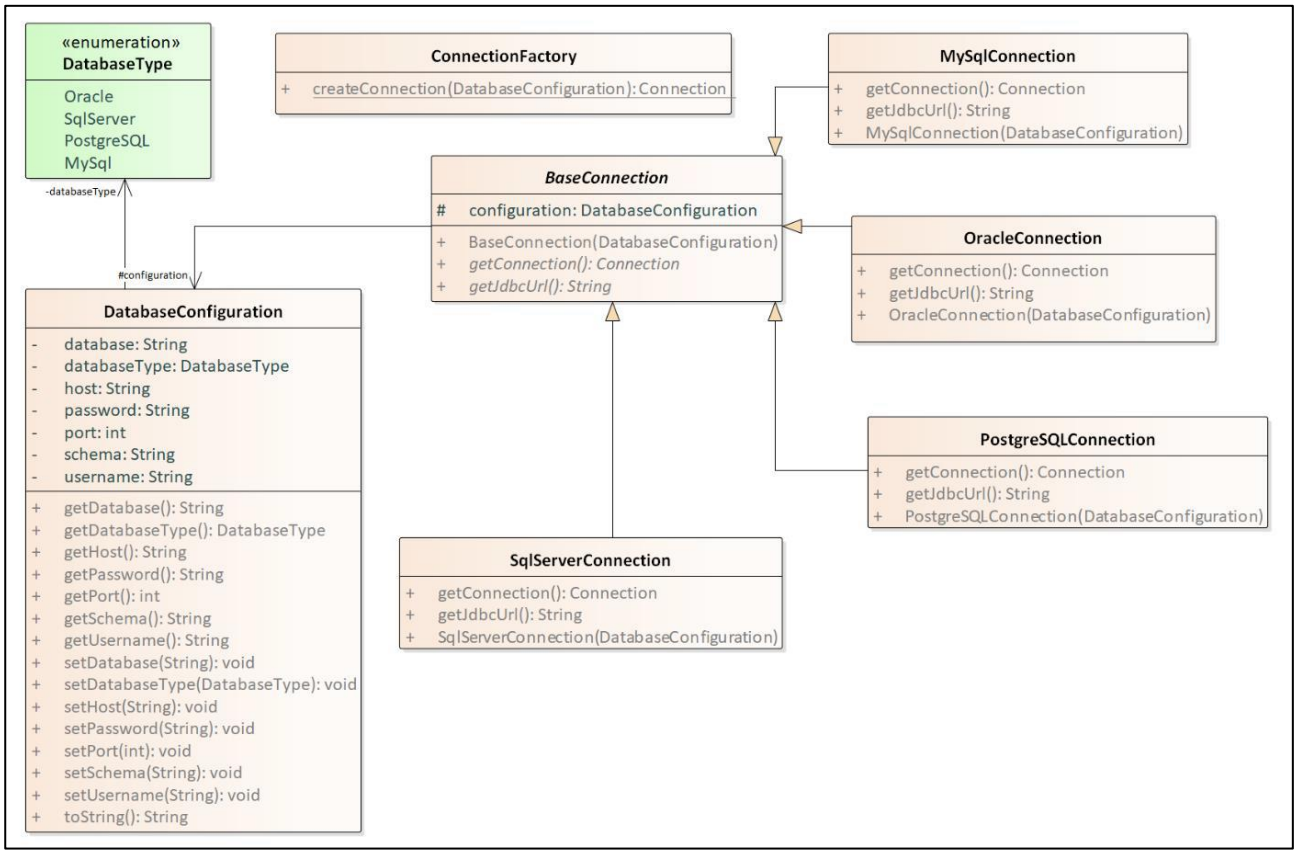
Desenvolvido por Aguiar (2019), o DB2JHipster é uma extensão do gerador de aplicações web chamado JHipster. Ele expande o comportamento tradicional do JHipster permitindo que o usuário gere as operações de Create, Read, Update and Delete (CRUD) para uma aplicação web a partir da conexão com uma base de dados. A conexão com o banco de dados é feita na tela inicial da ferramenta apresentada na Figura 2 (a). Após a conexão, uma barra de progresso é exibida, indicando o que está sendo carregado do banco, como as tabelas, atributos, chaves primárias e estrangeiras.

Figura 2 - Funcionamento do DB2JHipster

Fonte: adaptado de Aguiar (2019).

A Figura 3 apresenta o diagrama de classes do pacote `database`, o qual é responsável pelo processo que ocorre após clicar no botão `Avançar` da Figura 2 (a) e antes de chegar a tela da Figura 2 (b). O pacote contém as classes que fazem a conexão com o banco de dados. Não houve nenhuma alteração na estrutura desse pacote, apenas pequenos ajustes dentro de alguns métodos. A classe `DatabaseConfiguration` representa as configurações da base de dados, como nome da base, tipo do banco (representado pelo enumerador `DatabaseType`), servidor, porta, `schema`, usuário e senha. Para suportar diferentes tipos de banco (PostgreSQL, MySQL, SQL Server e Oracle), existe a classe abstrata `BaseConnection`, a qual é implementada pelas classes `PostgreSQLConnection`, `MySqlConnection`, `SqlServerConnection` e `OracleConnection`. A conexão realizada com o banco de dados ocorre a partir de uma instância do `DatabaseConfiguration`, armazenada na classe estática `ConnectionFactory`, conforme o tipo do banco.

Figura 3 - Diagrama de classes do pacote `database`



Fonte: Aguiar (2019, p. 31).

A Figura 2 (b) mostra a tela que permite alterar o tipo de relacionamento entre as tabelas, selecionando-se para isso uma entidade e escolhendo-se uma opção para cada item na lista de tipos de relacionamentos. Por fim, na Figura 2 (c) é exibido o código de especificação expresso em JDL gerado pela ferramenta a partir das tabelas, campos e

relacionamentos importados e/ou selecionados. O JDL pode ser editado pelo usuário, mas não existe nenhuma validação sobre o código alterado (AGUIAR, 2019, p. 47).

Após salvar o código JDL, o usuário precisa utilizar o *prompt* de comando para continuar a geração da aplicação pelo fluxo normal de funcionamento do JHipster. Inicialmente é necessário executar o comando `jhipster` e responder algumas perguntas para que o projeto inicial seja gerado (AGUIAR, 2019, p. 63). Então, após a geração do projeto concluída, o usuário precisa alterar o arquivo `application-dev.yml` com as configurações de acesso ao banco de dados alvo e a seguir utilizar o comando `import-jdl` para que o JHispter gere os artefatos necessários para levantar a aplicação. Quando o comando anterior estiver concluído, devem ser executados outros dois comandos, `mvnw` e `npm start`, respectivamente para iniciar o *backend* com Spring Boot e o *frontend* com o *framework* Angular (AGUIAR, 2019, p. 48-51).

Entre as limitações desta versão da ferramenta o Aguiar (2019, p. 63) cita que "[...] a principal limitação do protótipo é não aproveitar os dados da base existente usada como fonte para geração da aplicação web, pois o JHipster cria uma nova base e ignora os dados da base de referência.". Além disso, a ferramenta ignora campos não nulos, não leva em consideração o tamanho de campos do tipo *string* e não exibe uma descrição intuitiva para os relacionamentos das tabelas no *frontend* exibindo apenas o `id`. No entanto, Aguiar (2019, p. 63) conclui que "A principal contribuição da criação do protótipo é a possibilidade de escalar uma aplicação web a partir de um modelo de entidade relacionamento em pouco tempo, bem como na prototipação de uma nova aplicação que já tenha seu modelo definido.".

2.4 TRABALHOS CORRELATOS

O objetivo dessa seção é apresentar trabalhos com características semelhantes aos principais objetivos do estudo proposto. O Quadro 2 aborda uma ferramenta chamada Rizer (RIZER, 2020), o Quadro 3 traz alguns detalhes do W2PHP (SOMMARIVA, 2008) e a ferramenta FormGenerate (KLUG, 2007) é apresentada no Quadro 4.

Quadro 2 - Rizer

Referência	RIZER (2020)
Objetivos	Permitir que o usuário crie uma aplicação para a sua empresa de forma rápida e fácil por meio de cadastros sem requerer conhecimento em programação ou design.
Principais funcionalidades	a) gerar uma aplicação em Personal Home Page Hypertext Preprocessor (PHP) com banco de dados MySQL; b) criar tabelas e atributos por meio de cadastros, respectivamente chamados de módulos e campos pela ferramenta; c) definir validações para os campos criados, por exemplo: descrição, tipo do dado, valor padrão, obrigatoriedade e máscara; d) disponibilizar módulos prontos para a utilização, por exemplo: gestão de usuários, perfis de acesso, permissões, relatórios e indicadores; e) baixar o código-fonte da aplicação gerada.
Ferramentas de desenvolvimento	Embora tenha sido informado que a ferramenta gera aplicações em PHP, não foi possível identificar quais foram as ferramentas de desenvolvimento utilizadas para a criação dela.
Resultados e conclusões	Não foi possível testar a ferramenta, porém por meio de exemplos no próprio site da ferramenta foi possível verificar que as aplicações geradas são completas e possuem um *design* intuitivo.

Fonte: elaborado pelo autor.

Quadro 3 - W2PHP

Referência	SOMMARIVA (2008)
Objetivos	Permitir que o usuário gere operações de CRUD básico através do cadastro de tabelas e atributos.
Principais funcionalidades	a) gerar um CRUD básico, criar e conectar ele a uma base de dados; b) cadastrar as tabelas e os atributos da base de dados criada que serão gerados pelo CRUD; c) definir validações para os atributos, por exemplo: tipo e tamanho do dado, valor padrão, obrigatoriedade e preenchimento de zeros.
Ferramentas de desenvolvimento	PHP e MySQL.
Resultados e conclusões	A partir da análise de Sommariva (2008), percebe-se que o resultado do projeto permite a criação, por usuários leigos, de aplicações simples com dinamismo e interação com o banco de dados.

Fonte: elaborado pelo autor.

Quadro 4 - FormGenerate

Referência	KLUG (2007)
Objetivos	Possibilitar que o usuário efetue a criação de páginas Java Server Pages (JSP), a partir dos metadados de um banco relacional.

Principais funcionalidades	a) gerar páginas JSP no padrão MVC integradas com uma base de dados; b) realizar a leitura dos metadados de um banco relacional para obter tabelas e atributos; c) criar diferentes templates para visualização das telas de cadastro através de Velocity Template Language (VLT).
Ferramentas de desenvolvimento	Java, MySQL, Oracle e SQL Server.
Resultados e conclusões	Não foi possível testar a ferramenta, porém através dos exemplos apresentados na monografia foi possível verificar que ela proporciona uma economia de tempo aos usuários que já possuem uma base de dados criada. A ferramenta suporta várias opções de banco, em contrapartida, o usuário precisa de conhecimento prévio em VLT para construir os templates.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DA FERRAMENTA

Nesta seção são descritos os aspectos mais relevantes de especificação e implementação para a compreensão sobre o trabalho desenvolvido. Também são apresentados os requisitos e a explicação da operacionalidade da ferramenta.

3.1 REQUISITOS DO TRABALHO

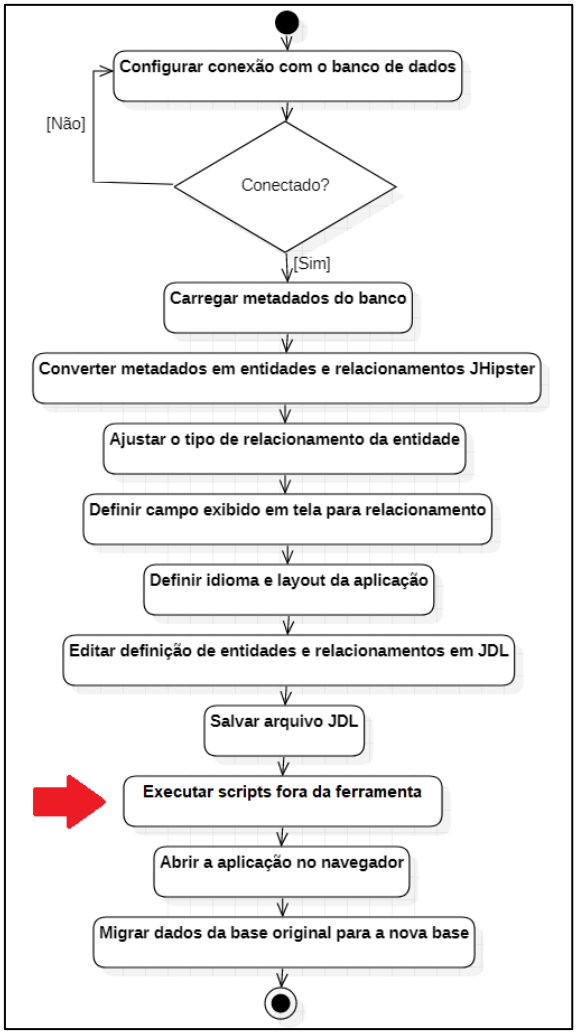
A ferramenta desenvolvida neste trabalho deve:

a) adequar-se às especificidades de modelos de dados dos bancos de dados: PostgreSQL, MySQL, SQL Server e Oracle (Requisito Funcional - RF);
b) validar os campos de aplicações geradas para garantir a integridade dos dados persistidos (RF);
c) permitir que o usuário defina a descrição de apresentação de campos que são chaves estrangeiras (RF);
d) adequar a ferramenta para interpretar as especificações das tabelas e os dados existentes (RF);
e) ativar o recurso de geração do código da aplicação web a partir da ferramenta eliminando a necessidade de execução de um programa externo (RF);
f) ser implementada em Java (Requisito Não Funcional - RNF);
g) utilizar o gerador JHipster (RNF).

3.2 ESPECIFICAÇÃO

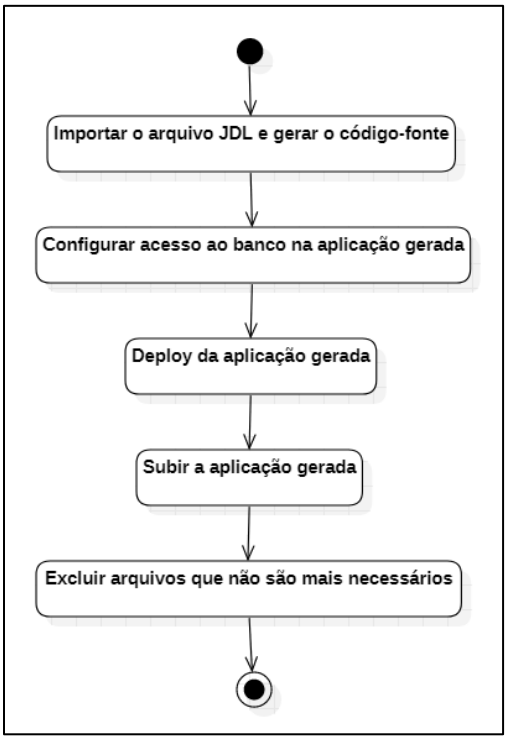
A estrutura e comportamento da ferramenta desenvolvida são apresentadas nesta seção por meio de diagramas Unified Modeling Language (UML). A Figura 4 exibe o diagrama de atividade da ferramenta, mostrando o fluxo de execução que ocorre dentro dela. A Figura 5 apresenta um diagrama de atividade com o fluxo fora da ferramenta (*prompt* de comando), o qual está relacionado com a atividade `Executar scripts fora da ferramenta` (sinalizado por uma seta vermelha na Figura 4). O fluxo se inicia com a configuração do banco de dados e encerra com a migração da base.

Figura 4 - Diagrama de atividade



Fonte: elaborado pelo autor.

Figura 5 - Executar scripts fora da ferramenta

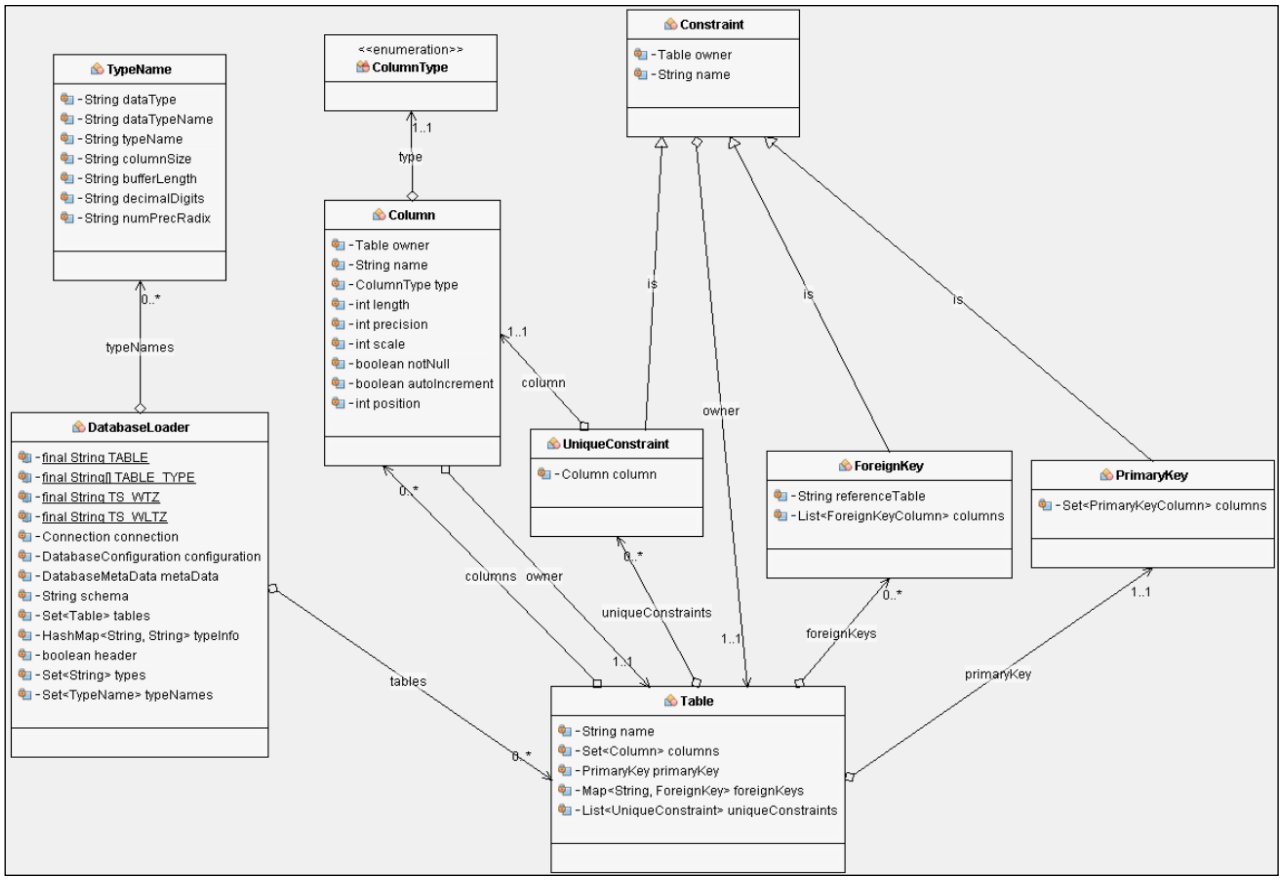


Fonte: elaborado pelo autor.

3.2.1 Diagrama de classes: pacote `metadata`

O pacote `metadata` é um dos mais importantes, ele contém as classes que armazenam as informações obtidas na leitura de metadados realizada na base do usuário. Muitas classes desse pacote se mantiveram da forma especificada por Aguiar (2019), embora os atributos praticamente não tenham sido alterados, vários métodos receberam novos tratamentos e alguns ajustes. A Figura 6 mostra o diagrama de classes atual da ferramenta (algumas classes e relacionamentos foram omitidos para facilitar a compreensão). A classe `DatabaseLoader` é responsável por realizar o processo de leitura dos metadados e guardar os objetos lidos no banco em suas respectivas classes, tabelas em `Table`, colunas em `Column`, chaves primárias, estrangeiras e únicas em `PrimaryKey`, `ForeignKey` e `UniqueConstraint`.

Figura 6 - Diagrama de classes do pacote `metadata`

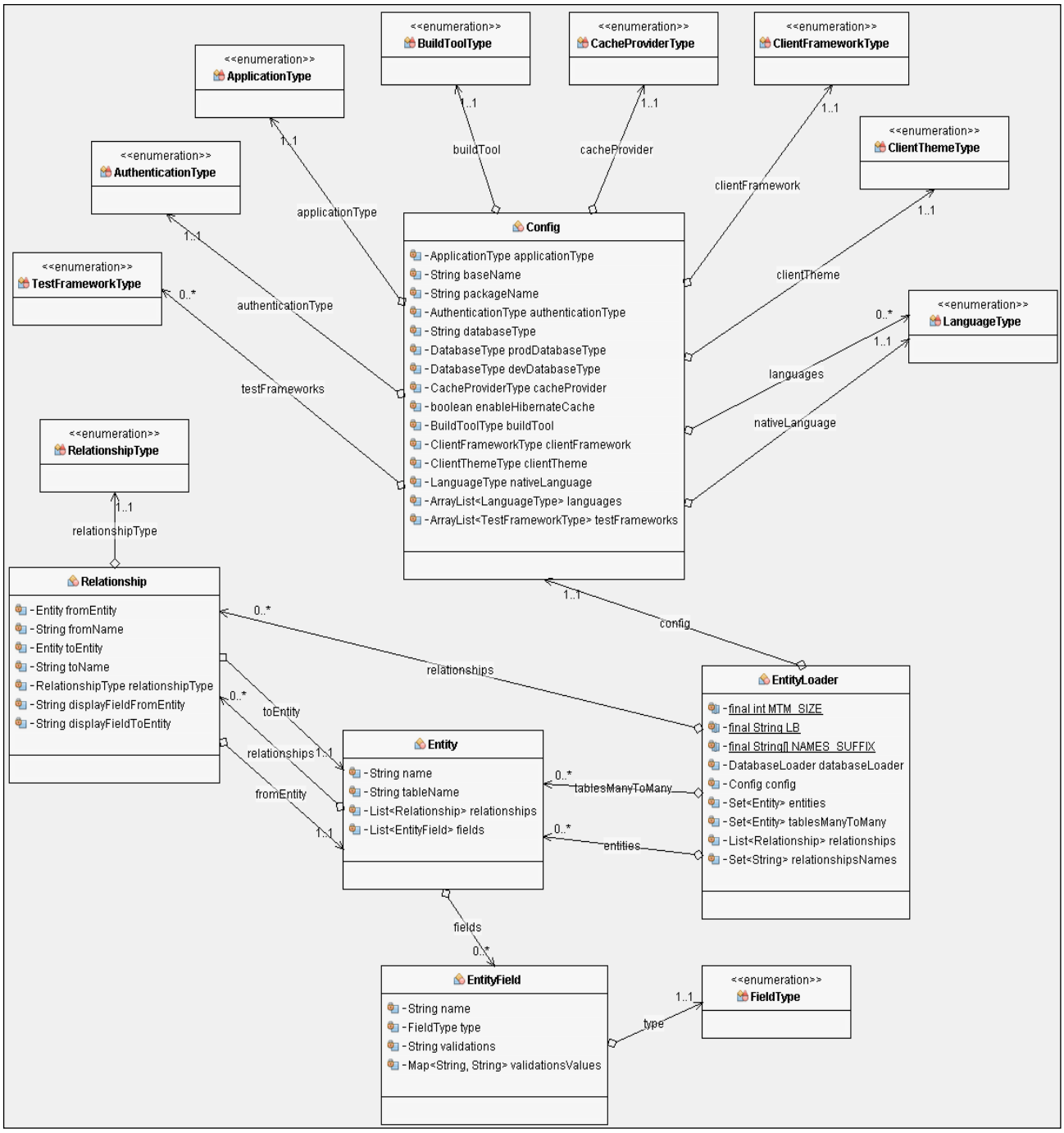


Fonte: elaborado pelo autor.

3.2.2 Diagrama de classes: pacote `jhipster`

O pacote `jhipster` agrega as classes que são utilizadas para montar as entidades e campos do JHipster com os seus devidos tipos. A maior mudança no pacote em comparação com o protótipo de Aguiar (2019) foi a criação da classe `Config` e outros enumeradores que definem suas propriedades. A Figura 7 traz a versão atual do diagrama de classes, novamente com os métodos omitidos para ajudar com o entendimento. A classe `EntityLoader` é a responsável pela conversão dos objetos armazenados nas classes do pacote `metadata`. Os objetos do tipo `Table` são convertidos em `Entity`, as colunas das tabelas se transformam em `EntityField` e os objetos dos diferentes tipos de chave em `Relationship`. As demais configurações informadas pelo usuário são guardadas na classe `Config`.

Figura 7 - Diagrama de classes do pacote `jhipster`



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nessa subseção serão destacados dois dos principais métodos utilizados pela ferramenta. O `loadTables` pertence ao pacote `metadata`, enquanto o `toJdl` ao pacote `jhipster`.

Quadro 5 - Método `loadTables` (Carregar tabelas)

```
131     /**
132      * Carrega as tabelas da base de dados
133      *
134      * @throws SQLException
135      */
136     public void loadTables() throws SQLException {
137         header = true;
138         loadTypeInfo();
139         tables.clear();
140         try (ResultSet result = getMetaData().getTables(null, null, null, TABLE_TYPE);) {
141             while (result.next()) {
142                 String tableSchema = result.getString("TABLE_SCHEM");
143                 String tableType = result.getString("TABLE_TYPE");
144                 if (isTable(tableType) && validateSchema(tableSchema)) {
145                     Table table = new Table(result.getString("TABLE_NAME"));
146                     tables.add(table);
147                 }
148             }
149         }
150     }
```

Fonte: elaborado pelo autor.

O método `loadTables` da classe `DatabaseLoader` apresentado no Quadro 5 é responsável por carregar as tabelas da base de dados. A leitura dos metadados é realizada por meio do método `getMetaData()`, pertencente à classe que implementa a interface `Connection` do banco utilizado. A variável `result` recebe o `getTables()`, o método se adequa ao banco e retorna as informações da tabela por meio das colunas `TABLE_SCHEM`, `TABLE_TYPE` e `TABLE_NAME`. As informações que ficam armazenados na variável `result` são validadas, geram instâncias de `Table` e então são inseridas em uma lista chamada `tables`.

Quadro 6 - Método `toJdl` (Converter para JDL)

```
188     public String toJdl() {
189         JdlWriter writer = new JdlWriter();
190         StringBuilder sb = new StringBuilder();
191
192         sb.append(writer.configToJdl(config));
193         sb.append(LB).append(LB);
194
            for (Entity entity : entities) {
196             sb.append(writer.entityToJdl(entity));
197             sb.append(LB).append(LB);
198         }
199
200         Map<RelationshipType, List<Relationship>> collect = relationships.stream()
201                 .collect(Collectors.groupingBy(Relationship::getRelationshipType));
202
203         collect.entrySet().stream().forEach(r -> {
204             sb.append(writer.relationshipToJdl(r.getKey(), r.getValue()));
205             sb.append(LB).append(LB);
206         });
207         return sb.toString();
208     }
```

Fonte: elaborado pelo autor.

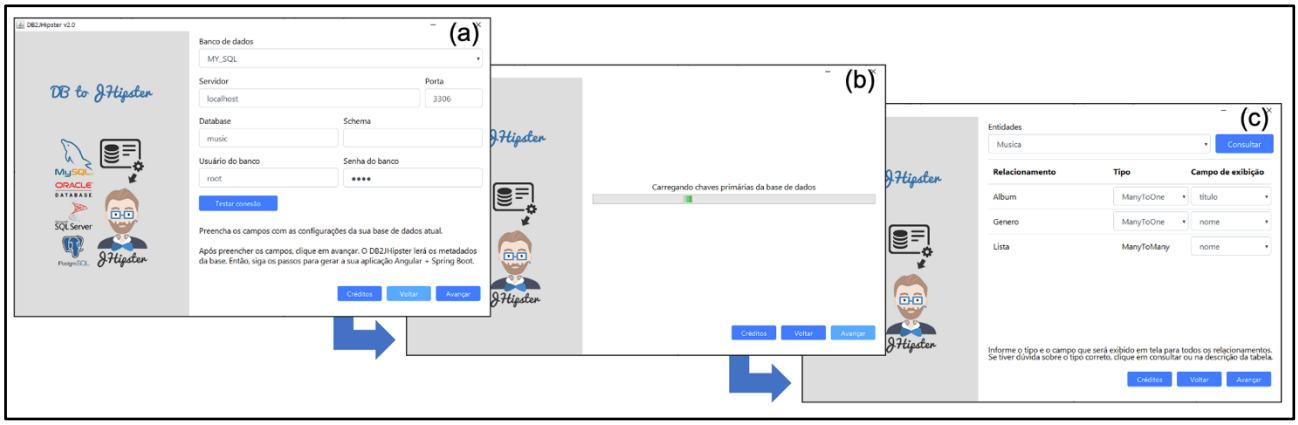
O conteúdo do arquivo JDL é gerado pelo método `toJdl` da classe `EntityLoader`, exibido no Quadro 6. É criada uma instância de `JdlWriter`, ela vai receber as configurações informadas pelo usuário e vai criar o cabeçalho para o arquivo, invocando o método `configToJdl()`. O cabeçalho criado é adicionado à um `StringBuilder` que armazenará todo o JDL. Então, inicia-se um laço de repetição, percorrendo todas as entidades JHipster geradas a partir das tabelas da base, uma a uma, as entidades são convertidas para o formato JDL por meio do método `entityToJdl()` e depois incluídas ao `StringBuilder`. De forma semelhante, os relacionamentos JHipster, vindos das chaves primárias e estrangeiras da base, são percorridos, convertidos para JDL e então, inseridos no `StringBuilder`. Por fim, o conteúdo do `StringBuilder` retorna como `String` para quem chamou o método `toJdl()`.

3.4 OPERACIONALIDADE DA FERRAMENTA

Para demonstrar o funcionamento da ferramenta desenvolvida, serão apresentadas as telas conforme o fluxo de execução. A Figura 8 (a) mostra a tela inicial em que são informadas as configurações para acessar o banco de dados. Na

Figura 8 (b) é apresentada a tela de carregamento que é exibida ao usuário, informando em qual estágio está o processo de leitura dos metadados da base informada. Durante o processo de leitura, a ferramenta cria objetos internos para armazenar toda a estrutura da base de dados, desde tabelas e colunas, até as chaves primárias e estrangeiras. O botão `Avançar` fica bloqueado enquanto a leitura está sendo executada e só é liberado após a carga completa da base. A terceira tela exibida no fluxo de execução da ferramenta está exposta na Figura 8 (c). Essa tela apresenta todas as tabelas da base ao usuário, ele deverá selecionar o tipo de relacionamento correspondente às tabelas e qual será o atributo exibido no lugar da chave primária e/ou estrangeira referente a ligação. É muito importante que o tipo de relacionamento seja escolhido corretamente, pois ele define a estrutura da aplicação que será gerada posteriormente. Se as informações de tipo não forem coerentes com a base de dados, a aplicação gerada poderá ter problemas ao inserir e/ou buscar dados nas tabelas. Para auxiliar o usuário com a correta seleção das informações, a ferramenta disponibiliza uma nova tela com uma consulta básica dos dados da tabela, conforme a Figura 9 em que se pode visualizar a maneira com que os dados estão dispostos na tabela. Esta nova tela pode ser acessada clicando-se no botão `Consultar` ou em cima do nome das tabelas relacionadas.

Figura 8 - Funcionalidades do DB2JHipster v2 – parte 1



Fonte: elaborado pelo autor.

Figura 9 - Consulta de dados da tabela selecionada

musica

id_musica	nome	album	genero	compositor
1574	Beth	126	1	Peter Criss, Stan Penridge, Bob Ezrin
1575	Nothin' To Lose	126	1	Gene Simmons
1576	Rock And Roll All Nite	126	1	Paul Stanley, Gene Simmons
1577	Immigrant Song	127	1	Robert Plant
1578	Heartbreaker	127	1	John Bonham/John Paul Jones/Robert Plant
1579	Since I've Been Loving You	127	1	John Paul Jones/Robert Plant
1580	Black Dog	127	1	John Paul Jones/Robert Plant
1581	Dazed And Confused	127	1	Jimmy Page/Led Zeppelin
1582	Stairway To Heaven	127	1	Robert Plant
1583	Going To California	127	1	Robert Plant
1584	That's The Way	127	1	Robert Plant
1585	Whole Lotta Love (Medley)	127	1	Arthur Crudup/Bernard Besman/Bukka White/Doc Pomus/John Bonham/John Lee Hooker/John Paul Jones/Mort Shuman/Robert Plant/Willie Dixon
1586	Thank You	127	1	Robert Plant
1587	We're Gonna Groove	128	1	Ben E.King/James Bethea
1588	Poor Tom	128	1	Jimmy Page/Robert Plant
1589	I Can't Quit You Baby	128	1	Willie Dixon
1590	Walter's Walk	128	1	Jimmy Page, Robert Plant
1591	Ozone Baby	128	1	Jimmy Page, Robert Plant
1592	Darlene	128	1	Jimmy Page, Robert Plant, John Bonham, John Paul Jones
1593	Bonzo's Montreux	128	1	John Bonham
1594	Wearing And Tearing	128	1	Jimmy Page, Robert Plant

Fonte: elaborado pelo autor.

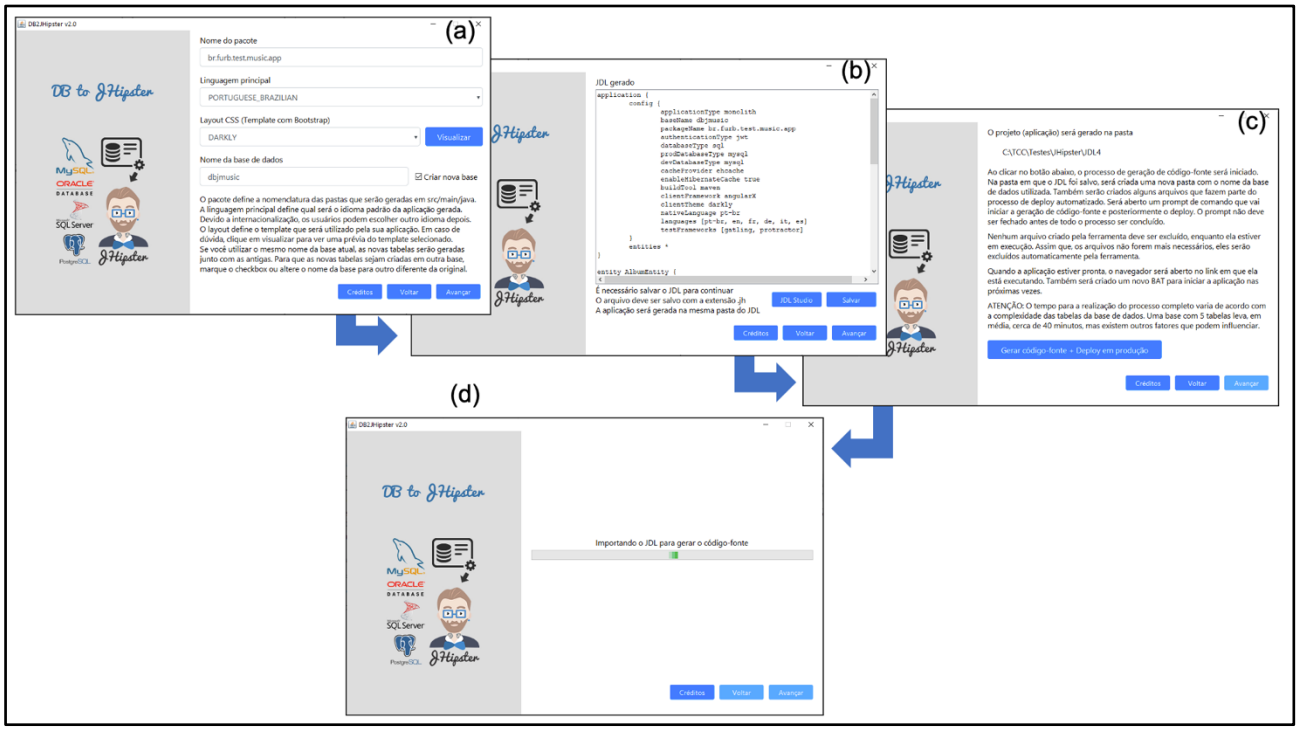
A tela apresentada na Figura 10 (a) mostra algumas configurações que podem ser feitas para a aplicação que será gerada. É possível informar a nomenclatura de pastas a ser utilizada dentro de `src/main/java`. O usuário pode selecionar o idioma padrão em `Linguagem principal`, mas devido ao conceito de internacionalização aplicado, posteriormente, cada usuário da aplicação gerada poderá escolher o idioma que desejar. Também é possível escolher o *template* utilizado. Para auxiliar o usuário na escolha do layout, ele pode clicar em `Visualizar` para ver uma prévia diretamente no navegador. Por fim, existe um campo para informar o nome da base de dados. Caso seja utilizado o mesmo nome da base informada na tela inicial para a leitura dos metadados, as novas tabelas serão geradas junto com as antigas, caso contrário uma nova base de dados será criada sem afetar a estrutura e os dados da base original.

O código JDL gerado é apresentado na Figura 10 (b), de forma que um usuário mais avançado pode realizar alguma edição adicional que julgue necessária (contudo, alterações não são recomendadas, visto que não existe uma validação sobre o código editado). Avançando no processo, a especificação JDL deve ser salva, desbloqueando assim o botão `Avançar`. É importante que o usuário leve em consideração, no momento de escolher o local em que o arquivo será salvo, que a aplicação será gerada na nessa mesma pasta.

Conforme a Figura 10 (c), a próxima tela exibe o local em que a aplicação vai ser gerada e traz uma breve explicação referente ao processo de geração do código-fonte. Para continuar e iniciar o processo de geração é necessário

clicar em `Gerar código-fonte + Deploy em produção`. Após clicar no botão, alguns arquivos serão criados na pasta escolhida e o usuário vai ser redirecionado para uma nova tela, ela pode ser vista na Figura 10 (d).

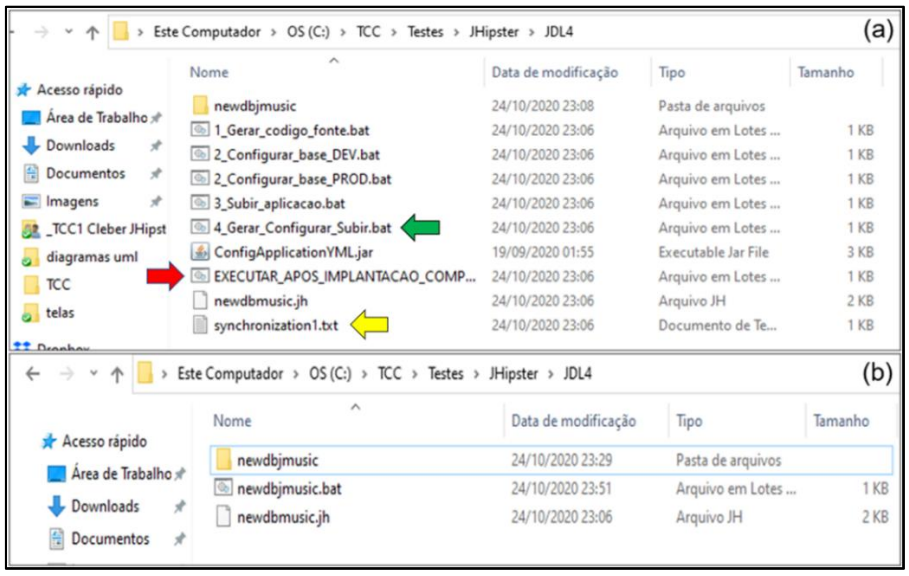
Figura 10 - Funcionalidades do DB2JHipster v2 – parte 2



Fonte: elaborado pelo autor.

Inicialmente a pasta escolhida tinha apenas um arquivo JDL, mas depois dos novos arquivos gerados, ela fica da forma que pode ser vista na Figura 11 (a). Os arquivos são de diferentes formatos pois existem aqueles com formato .BAT (arquivos de lotes do Windows) que são utilizados para poupar o usuário do trabalho de configurar manualmente a aplicação em linha de comando. Estes arquivos estão numerados na sequência em que precisam ser ativados. Neste sentido, o arquivo `4_Gerar_Configurar_Subir.bat` (sinalizado por uma seta verde na Figura 11 (a)) é chamado a partir da ferramenta, o *prompt* de comando é aberto e outros arquivos de lotes são chamados de maneira síncrona, ou seja, um novo arquivo de lote só inicia após o término do outro. Logo nos primeiros comandos, uma subpasta com o nome da base é criada, dentro da qual o código-fonte será gerado. Para que o status exibido na ferramenta seja atualizado conforme a execução de comandos, alguns arquivos de sincronização são criados na pasta, por exemplo: `synchronization1.txt` (sinalizado por uma seta amarela na Figura 11 (a)).

Figura 11 - Arquivos gerados na pasta

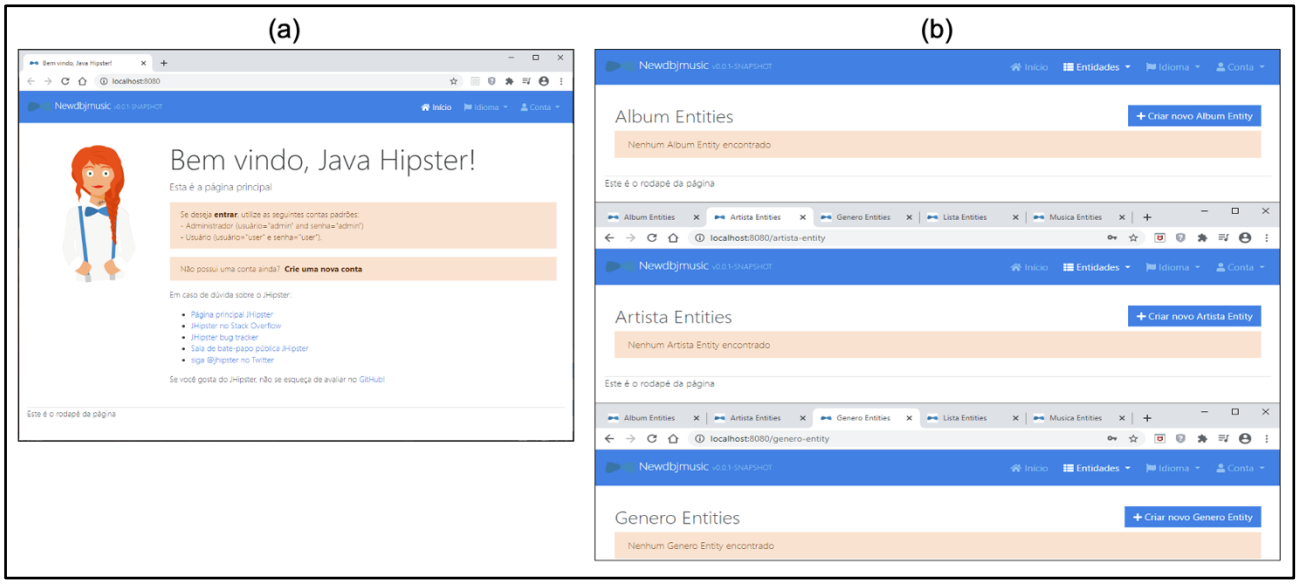


Fonte: elaborado pelo autor.

Após a execução de todos os arquivos de lotes, o navegador é aberto com a aplicação rodando (Figura 12 (a)). Neste momento como os arquivos temporários já não são mais necessários, a ferramenta chama um último arquivo denominado `EXECUTAR_APOS_IMPLANTACAO_COMPLETA.BAT` (sinalizado por uma seta vermelha na Figura 11 (a)) limpando a pasta e substituindo o seu conteúdo por um único arquivo de lote que permite a ativação da aplicação quantas vezes for desejado (Figura 11 (b)).

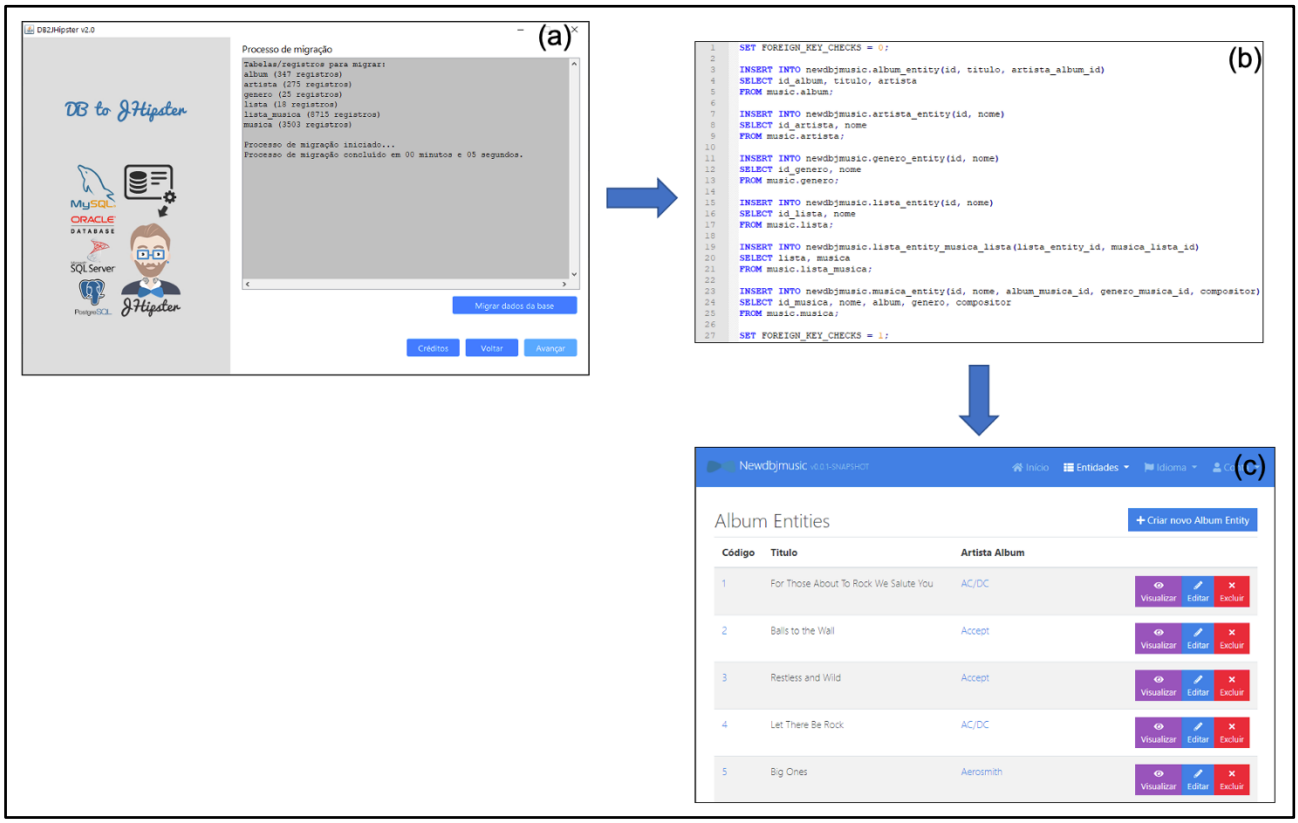
Depois de criar um usuário e realizar o login na aplicação, é possível navegar entre as entidades geradas (Figura 12 (b)) observando-se que nenhuma entidade possui registros. Assim fica a critério do usuário realizar ou não a migração da base original. Caso o usuário deseje realizar a migração dos dados da base original, ele pode voltar para ferramenta e `Avançar` para a tela que está na Figura 13 (a), caso contrário ele pode simplesmente fechar a ferramenta e começar a usar a aplicação com a base limpa.

Figura 12 - Aplicação rodando no navegador



Fonte: elaborado pelo autor.

Figura 13 - Pasta após conclusão do processo



Fonte: elaborado pelo autor.

Caso o usuário da ferramenta tenha optado por migrar a base original, conforme mencionado no parágrafo anterior, ele é redirecionado para a tela que está na Figura 13 (a). Para iniciar o processo de migração é necessário clicar em `Migrar dados da base`, então um script SQL (Figura 13 (b)) é criado dentro da pasta em que a aplicação foi gerada. Depois de ser criado ele é executado sobre banco de dados da aplicação migrando todas as tabelas e atributos da base original (nesta versão não é possível filtrar os dados que serão importados). Ao retornar para o navegador e recarregar a página, os dados migrados são apresentados (Figura 13 (c)).

4 RESULTADOS

Ao final do trabalho, considera-se que os requisitos propostos foram atendidos. Conforme apresentado, o presente trabalho descreveu os melhoramentos incorporados na versão inicial da ferramenta DB2JHipster de Aguiar (2019), facilitando o processo de conexão com as bases de dados selecionadas. Além disso, introduziu uma camada de validação dos relacionamentos identificados no modelo de dados, permitindo qualificá-los adequadamente. Desse modo, considera-se que os objetivos foram atendidos. O Quadro 7 apresenta um comparativo da ferramenta com os trabalhos correlatos e o protótipo criado por Aguiar (2019).

Quadro 7 - Comparativo com correlatos e versão anterior

Trabalhos / Características	Rizer (2020)	Sommariva (2008)	Klug (2007)	Aguiar (2019)	DB2JHipster v2.0
Geração de telas de CRUD	X	X	X	X	X
Funciona com diferentes bancos			X	X	X
Importa dados da base existente			X		X
Adiciona máscaras nos campos	X				
Linguagem utilizada	PHP	PHP	Java	Java	Java

Fonte: elaborado pelo autor.

A partir da análise dos correlatos, conclui-se que a ferramenta DB2JHipster versão 2 atende ao requisito de agilizar o desenvolvimento de uma nova aplicação web a partir de tabelas de um banco de dados existente. Apesar das diferenças existentes entre eles, todos geram algum tipo de formulário que permite inserir, editar e remover os dados de uma base. Com relação ao banco de dados utilizado, o Rizer (2020) e o Sommariva (2008) funcionam apenas com banco MySQL, enquanto o Klug (2007) possibilita que o usuário utilize também Oracle e SQL Server, mas somente o DB2JHipster (AGUIAR, 2019) permite além destes, a utilização de PostgreSQL. Apenas o Rizer (2020) permite que o usuário selecione máscaras de dados pré-configuradas, como por exemplo: telefone, e-mail, CPF, número, entre outros. O DB2JHipster (AGUIAR, 2019) e o FormGenerate (KLUG, 2007) são os únicos que criam o projeto utilizando dados de bases pré-existentes. Conforme foi comentado, todos os correlatos além do DB2JHipster possuem algumas características diferentes e nenhum deles atendeu completamente a todos os pontos avaliados, pois ainda que similares, eles foram criados com objetos específicos diferentes.

Para verificar a evolução da ferramenta com relação ao trabalho de Aguiar (2019), foi realizado um teste utilizando-se a mesma base de dados em ambas as versões. A base utilizada para o teste continha dados relacionados com músicas. O resultado pode ser visto na Figura 14 (os registros exibidos pertencem a entidade `Lista`) em que fica evidenciado na parte superior da figura que na versão anterior somente o `id` das chaves primárias era exibido, enquanto na parte inferior é exibido o atributo escolhido pelo usuário. Outro ponto a ser destacado é que a geração do código-fonte e migração de dados da base de origem eram realizados manualmente na versão de Aguiar (2019) (através de operações em linha de comando e criação/execução de script no banco) e na versão atual, a ferramenta absorveu estas funcionalidades e o faz de forma transparente para o usuário (geração de código-fonte e a migração de dados).

Figura 14 - Comparativo entre versões do projeto DB2JHipster

Fonte: elaborado pelo autor.

5 CONCLUSÕES

Esse trabalho apresentou avanços no desenvolvimento da versão inicial do projeto DB2JHipster aproximando-a do objetivo de torná-la efetivamente uma ferramenta prática. Apesar de ter iniciado o projeto a partir dos fontes da versão original, surgiram problemas relacionados com o download de dependências do Maven impossibilitando a execução da versão 1 diretamente. Objetivando superar esta dificuldade, foi iniciado um novo projeto no NetBeans (a versão anterior utilizou Eclipse) sem o uso do Maven realizando-se a migração das classes e pacotes para o novo projeto. Esta estratégia viabilizou a organização do projeto no novo ambiente de desenvolvimento eliminando a questão das dependências. A partir da leitura dos metadados de uma base utilizada como entrada, a versão 2 apresentou opções de configurações e depois gerou o JHipster Domain Language (JDL) com a estrutura do banco convertida para JHipster. Depois do JDL gerado, a ferramenta trouxe outras telas para que fosse realizado o *deploy* automatizado e a migração da base original.

A principal limitação identificada nesta versão está associada ao processo de identificação de relacionamentos `ManyToMany` no momento da leitura dos metadados, pois neste caso a tabela obrigatoriamente precisa ter apenas dois atributos, sendo que ambos são ao mesmo tempo chaves primárias e secundárias. Caso o usuário informe uma base em que a tabela não está nesse padrão, a versão 2 não consegue classificar os atributos adequadamente.

Em função dos aspectos acima apresentados é possível concluir que os objetivos estabelecidos no projeto foram alcançados e a eficácia da versão 2 foi validada por meio de testes comparativos com a versão 1. Assim o presente projeto contribui cientificamente por permitir a aceleração do início de projetos de pequeno e médio portes, reduzindo o tempo gasto com a criação de *mockups* e facilitando a elaboração de provas de conceito para projetos de longo prazo ou complexidade elevada. Do ponto de vista tecnológico, a ferramenta construída utiliza um *framework* de última geração, cujas funcionalidades vêm sendo utilizadas por grandes empresas[1]. A ferramenta também contribui sob o ponto de vista social, na medida em que poderá ser utilizada em sala de aula para apoiar projetos de disciplinas como Projeto de Software I e II e Trabalho de Conclusão de Curso, por exemplo.

5.1 EXTENSÕES

Sugere-se como extensões da ferramenta apresentada:

a) pesquisar estratégias para melhorar a funcionalidade de identificação de relacionamentos `ManyToMany`;
b) permitir que sejam adicionadas máscaras nos campos através da ferramenta, utilizando a validação `pattern` do próprio JHipster;
c) adaptar a versão 2 permitindo a anonimização dos dados sensíveis no processo de migração da base de dados;
d) permitir que seja definido o tipo de busca dos dados como `LAZY` ou `EAGER` nos relacionamentos;
e) gerar código-fonte para outras tecnologias, por exemplo: diretamente para o Spring Boot.

REFERÊNCIAS

ABREU, Bernardo Gomes de. **Desenvolvimento de um sistema Web para utilização e gerenciamento de dados de Cupons Fiscais e Saúde**. 2016. 84 f. TCC (Graduação) - Curso de Sistemas de Informação, Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade, 2016.

[1] https://www.jhipster.tech/companies-using-jhipster/

AGUIAR, Ingmar Schmidt de. **Protótipo de um gerador de aplicações web com JHipster**. 2019. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Universidade Regional de Blumenau, Blumenau, 2019.

BETTENBURG, Nicolas; JUST, Sascha; SCHRÖTER, Adrian; PREMRAJ, R.; ZIMMERMANN, T.. What Makes a Good Bug Report? In: International Symposium on Foundations of software engineering '08/FSE-16, 16., 2008, Atlanta. **Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering**. Atlanta: Acm, 2008. v. 1, p. 308-318.

DUBOIS, Julien; SASIDHARAN, Deepu K; GRIMAUD, Pascal. **Greetings, Java Hipster!**. 2020. Disponível em: https://www.jhipster.tech. Acesso em: 14 jun. 2020.

HALIN, Axel; NUTTINCK, Alexandre; ACHER, Mathieu; DEVROEY, Xavier; PERROUIN, Gilles; BAUDRY, Benoit. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. **Empirical Software Engineering**, [s.l.], v. 24, n. 2, p. 674-717, 17 jul. 2018. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s10664-018-9635-4. Disponível em: https://doi.org/10.1007/s10664-018-9635-4. Acesso em: 14 jun. 2020.

JHIPSTER. **JHipster**. [S. l.], 2020. Disponível em: https://www.jhipster.tech/. Acesso em 23 mar. 2020.

JIN, Dongpu; QU, Xiao; COHEN, Myra B.; ROBINSON, Brian. Configurations everywhere: implications for testing and debugging in practice. In: International conference on software engineering 2014, 36., 2014, Hyderabad. **Proceedings of the 36th international conference on software engineering**. New York: Association For Computing Machinery, 2014. v. 1, p. 215-224. Disponível em: http://dx.doi.org/10.1145/2591062.2591191. Acesso em: 14 jun. 2020.

PRATES JUNIOR, Lazaro. **Introdução ao Framework JHipster**. [S. l.], 2016. DevMedia. Disponível em: https://www.devmedia.com.br/introducao-ao-framework-jhipster/34043. Acesso em: 07 jun. 2020.

KLUG, Maicon. **Gerador de Código JSP Baseado em Projeto de Banco de Dados**. 2007. 121 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Universidade Regional de Blumenau, Blumenau, 2007.

LIMA, Carlos Filipe da Silva. **Full Stack Application Generation for Insurance Sales based on Product Models**. 2016. 165 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Informática, Instituto Superior de Engenharia do Porto, Porto, 2016. Disponível em: https://recipp.ipp.pt/bitstream/10400.22/10829/1/DM_CarlosLima_2016_MEI.pdf. Acesso em: 4 jun. 2020.

MACHADO, Nielsen Luiz Rechia. **Um framework para identificação e monitoramento de perfis e comportamentos de consumidores baseado no uso de aplicativos em dispositivos móveis**. 2019. 205 f. Tese (Doutorado) - Curso de Programa de Pós-graduação em Ciência da Computação, Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2019.

MATTSSON, Michael. **Object-Oriented Frameworks**: a survey of methodological issues. 1996. 129 f. Thesis, Department of Computer Science and Business Administration, University College of Karlskrona/Ronneby, Ronneby, 1996.

MUNDIM, Camila Augusto; SIESTRUP, Julia Fialho Grosse. **Gerenciamento estratégico da transformação digital**: Perspectivas conceituais e estudo de caso de uma grande empresa petrolífera. 2019. 106 f. Projeto (Graduação) - Curso de Engenharia de Produção, Escola Politécnica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2019.

NASCIMENTO, Leonardo Amaro do. **Softwares, Sistemas e Aplicações**: a evolução no mundo das soluções. [S. l.], 2020. Disponível em: https://administradores.com.br/artigos/softwares-sistemas-e-aplica%C3%A7%C3%B5es-a-evolu%C3%A7%C3%A3o-no-mundo-das-solu%C3%A7%C3%B5es. Acesso em: 11 abr. 2020.

RAIBLE, Matt. **The JHipster mini-book**. Birmingham: C4Media, 2019. 170 f. ISBN: 978-1-329-63814-3. Disponível em: https://www.infoq.com/minibooks/jhipster-mini-book-5/. Acesso em: 17 abr. 2020.

RED HAT. **O que é transformação digital e como alcançá-la**. [S. l.], 2017. Disponível em: https://www.redhat.com/pt-br/topics/digital-transformation/what-is-digital-transformation. Acesso em: 2 jun. 2020.

RIZER. **Rizer**. [S. l.], 2020. Disponível em: https://www.rizer.com.br/. Acesso em: 12 abr. 2020.

SILVA NETO, José Antonio; VILAR, Rodrigo. **Geração de código baseada em metamodelos para segurança em sistemas de informação web**. 2019. 14 f. TCC (Graduação) - Curso de Sistemas de Informação, Departamento de Ciências Exatas, Universidade Federal da Paraíba, Rio Tinto, 2019.

SOMMARIVA, Leonardo Waltrick. **Ferramenta Visual para Geração de Arquivos de Script em PHP**. 2008. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Universidade Regional de Blumenau, Blumenau, 2008.

VOLPON, Flávio Henrique Fernandes. **Simulação do fluxo de peças durante a operação de torneamento em sistemas flexíveis de fabricação baseado em framework orientado a objetos**. 2011. 123 f. Dissertação (Mestrado em Manufatura) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.