WEB FORMS – PROTÓTIPO DE UMA FERRAMENTA DE CONSTRUÇÃO DE FORMULÁRIOS E COMPONENTIZAÇÃO PARA REACT

Alexandre Leonardo Romero, Mauro Mattos – Orientador

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

alexandre.romero@furb.br, mattos@furb.br

Resumo: *A pandemia pela qual o mundo passa tem promovido uma forte pressão pela digitalização de negócios o que tem sido chamado de transformação digita. Este movimento gera a necessidade de migração de aplicações legadas para o ambiente web para permitir uma interação mais rápida entre o cliente e o fornecedor. Há várias alternativas disponíveis no mercado que passam por ferramentas denominadas low-code (quando o ambiente gera uma aplicação automaticamente com mínimo esforço do desenvolvedor). Este artigo descreve o desenvolvimento de um protótipo de uma ferramenta web para agilizar a criação de formulários baseados na tecnologia React na perspectiva de contribuir para a diminuição da curva de aprendizagem desta tecnologia. São apresentados a especificação, um exemplo de funcionamento e os resultados obtidos que demonstram que os objetivos foram atingidos*.

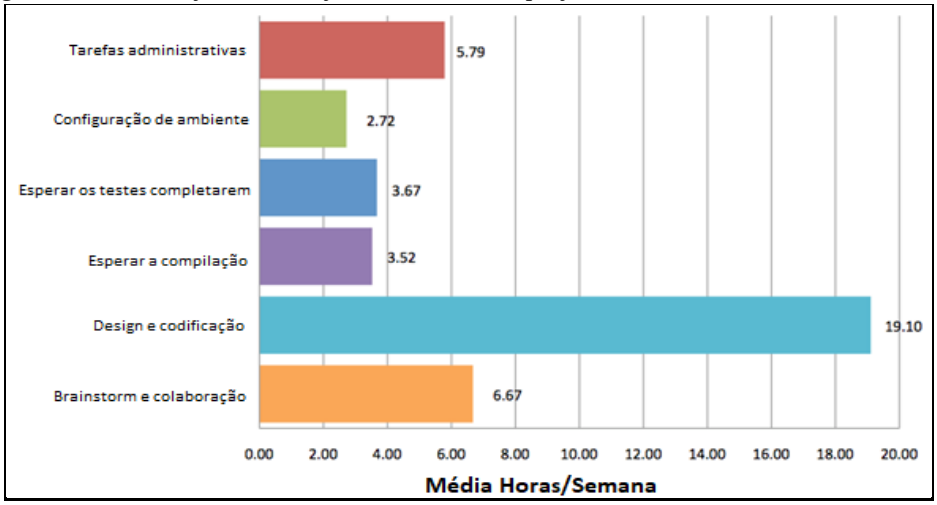
Palavras-chave*: Transformação digital. Formulários web. React. Low-code.*

1 INTRODUÇÃO

Num cenário de transformação digital que impacta todo um ecossistema envolvendo clientes, funcionários e empresários, os ambientes de negócios digitais são caracterizados por maior interconectividade e expansão de interdependências em que a incerteza e o dinamismo nos ambientes de mercado digitalizados exigem uma maior agilidade das estratégias de negócios (TEECE; PETERAF; HEATON, 2016, p. 1). Os conceitos estratégicos e as configurações organizacionais existentes devem ser revisados devido ao papel abrangente da TI que evolui de um mero suporte de negócios para uma parte integrante de negócios inteiros (WUNDERLICH, 2018, p. 2). Neste contexto, os entregáveis em termos de software demandam novas perspectivas, utilizam novas metodologias e culturas de trabalho. Os requisitos são cada vez mais superficiais, escondendo por vezes a dificuldade no entendimento e possíveis alterações e os *deadlines* são cada vez mais apertados com necessidade da entrega com excelência em pouco tempo para desenvolver (GAEA, 2019, p. 1).

Conforme Transformação Digital (2017, p. 1), "o crescente destaque da computação em nuvem desperta nos fornecedores de software o interesse em ganhar posição competitiva, bem como motiva os clientes a mudarem suas estratégias de TI para a nuvem". Assim, criar ou redesenhar uma aplicação web, independente da tecnologia, pode não ser uma tarefa tão simples (GONÇALVES, 2019). Contudo, uma das maiores dificuldades no desenvolvimento de aplicações web está associada ao desenvolvimento da chamada camada de *frontend*, geralmente associada a construção de formulários para entrada de dados e interação com o usuário. Neste contexto, Krill (2013) apresenta alguns dados que caracterizam este problema e é ilustrado através da Figura 1.

Figura 1 – Distribuição do esforço semanal em um projeto de desenvolvimento de software.



Fonte: Krill (2013, p. 1).

Em função desta realidade, Rivero *et al.* (2014, p. 670-687) afirmam que enfoques de desenvolvimento ágeis estão se tornando um padrão na indústria de desenvolvimento de aplicações *web* e as Model-Driven Web Engineering (MDWE) têm demonstrado melhorar a produtividade na construção destes tipos de aplicações. As MDWE direcionam maior atenção no processo de modelagem, em relação ao processo de codificação e na derivação de uma aplicação executável através da realização de transformações, a partir de modelos conceituais em código executável (ROSSI *et al.*, 2016, p. 1).

Diante dos argumentos citados, o presente projeto descreve o processo de construção de um protótipo de uma ferramenta que possibilita a especificação e a renderização de interfaces *web*, baseadas em formulários de entrada de dados, a partir de especificações em notação Java Script Object Notation (JSON) e a geração de código fonte para componentes React. Para isso, foi utilizado o *framework* PHP Yii e Javascript para conversão do JSON em componente React.

Os objetivos do trabalho são:

a) conceber um modelo de especificação de formulários em notação JSON;
b) conceber uma arquitetura de renderização em navegadores web que permita transformar as especificações em modelos executáveis;
c) realizar um conjunto de testes funcionais para avaliar preliminarmente a solução construída.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem por objetivo apresentar dois dos principais assuntos que estão relacionados com o trabalho. A subseção 2.1 aborda o Model Driven Web Engineering (MDWE), enquanto a subseção 2.2 trata de Geradores de Código e finalmente a subseção 2.3 trata sobre Linguagens de Domínio Específico (DSL).

2.1 MODEL DRIVEN WEB ENGINEERING

O crescimento explosivo da *web* como plataforma para a construção de aplicativos de software e como meio de comunicação, entretenimento, educação e comércio mudou drasticamente o cenário do desenvolvimento de software Rossi *et al*. (2016, p. 1). Neste contexto, a gestão dos artefatos de software tem aumentado ao longo do tempo em complexidade, elevando naturalmente o nível de esforço necessário para a sua produção. Esta é uma das grandes questões que a Engenharia de Software vem tratando nas últimas décadas e neste sentido Whittle, Hutchinson e Roucefield (2014, p. 39) afirmam que a Model Driven Engineering (MDE) está associada ao "[...] uso sistemático de modelos como artefatos primários durante um processo de Engenharia de Software". Já Oliveira (2012, p. 42), afirma que é "[...] uma abordagem de desenvolvimento de *software* que se concentra na criação de modelos que descrevem os elementos de um sistema e orientam sua implementação".

> As abordagens MDWE têm como objetivo melhorar o desenvolvimento de softwares Web, tornar a modelagem algo mais importante que a codificação e fazer dos modelos criados código em tempo de execução. O surgimento da Linguagem de Modelagem de Fluxo de Interação (IFML) é considerado um marco na evolução das linguagens de modelagem, indicando a maturidade da área e uma convergência de linguagens. Então a MDWE é uma especialização da área de Engenharia de Software Orientada a Modelos (MDSE) que é uma abordagem de desenvolvimento de software baseado em modelos que geram código fonte em tempo de execução, mas para área de aplicativos Web. (ROSSI *et al*. 2016, p. 1).

Rossi *et al*. (2016, p. 1) citam como exemplos deste paradigma os modelos de aplicativos Web indicando que "[...] uma das muitas facetas do MDWE foi o desenvolvimento de linguagens de modelagem da Web, aquelas que permitem a construção de modelos de aplicativos da Web", e concluem indicando que os modelos MDWE têm demonstrado uma clara separação de papéis, favorecendo a modularização de código e, como consequência, a sua evolução.

2.2 GERADORES DE CÓDIGO

Conforme Klug (2007), geradores de código são mecanismos auxiliares ao processo de criação de um software e surgem com objetivo de reduzir tempo e custos de desenvolvimento, bem como ganho de produtividade. A partir de uma entrada formatada em algum protocolo predefinido são gerados os arquivos fonte de uma aplicação completa ou artefatos específicos. O uso desta tecnologia é empregado principalmente na ausência de tempo para o desenvolvimento e/ou quando a equipe para implementação do projeto é pequena (KLUG, 2007, p.18). Segundo Herrington (2003, p. 3, tradução nossa) a "geração de código trata sobre escrever programas que escrevem programas". Em outras palavras, tal ato é uma técnica para se construir código utilizando um ou mais programas (KLUG, 2007). As técnicas de geração de código fornecem benefícios substanciais aos engenheiros de software em todos os níveis, os quais, segundo Herrington (2003, p.15) incluem:

a) qualidade e consistência: a geração de código a partir de modelos cria instantaneamente uma base de código mais consistente, consequentemente com mais qualidade;
b) único ponto de conhecimento: usando um gerador de tabelas de banco de dados como exemplo, um código manual, para alterar o nome de uma tabela, seria necessário alterar todos os códigos que possuam tal tabela manualmente, com um gerador de código basta alterar num ponto e o restante fica sob sua responsabilidade;
c) mais tempo para o projeto, agilidade: a geração de código reduz significativamente o tempo de desenvolvimento, permitindo reduzir prazos, ou até mesmo ganhar mais tempo para outras etapas do projeto.

2.3 LINGUAGEM DE DOMÍNIO ESPECÍFICO

Para Voelter (2013), uma Linguagem de Domínio Específico (DSL) é uma linguagem de programação de computadores restrita, focada em um domínio (problema) específico e, ao contrário das tradicionais linguagens de programação de uso mais geral, uma DSL possui sintaxe e semântica delimitadas no mesmo nível de abstração que o domínio oferece. No entanto como aponta Ghosh (2011, p. 10), programas escritos usando uma DSL devem ter as mesmas qualidades esperadas em um programa escrito em qualquer outra linguagem de programação. Em outras palavras, uma DSL é puramente uma linguagem otimizada para um determinado tipo de problema, chamado domínio e baseia-se em abstrações estreitamente alinhadas com o domínio (problema) para qual foi construída (VOELTER, 2013).

As DSLs podem ser classificadas em dois grupos: interna e externa. Uma DSL do tipo interna é aquela que usa a estrutura de uma linguagem de programação existente, como por exemplo o Rails, que é implementada sobre a linguagem de programação Ruby. Já uma DSL externa precisa ser desenvolvida e ter uma infraestrutura separada para análise léxica, técnicas de análise, interpretação, compilação e geração de código (VOELTER, 2013). Dentre os motivos para se usar uma DSL pode-se citar a aprimoração da produtividade, quanto mais fácil de ler um trecho de código, mais fácil de entendê-lo e consequentemente encontrar erros e modificá-lo (MASCARENHAS, 2017).

Mascarenhas (2017) ainda destaca alguns problemas que podem surgir com o uso de DSL, como, por exemplo, o uso de muitas linguagens no mesmo projeto dificulta a adesão de novas pessoas ao trabalho além do custo associado ao seu desenvolvimento e a sua manutenção.

2.4 REACT

O React é um *framework* Javascript criado por engenheiros do Facebook para resolver problemas envolvidos na construção de complexas interfaces com o usuário envolvendo conjunto de dados dinâmicos e envolve o domínio por parte do desenvolvedor do seguinte conjunto de tecnologias para funcionar: *javascript*, Hyper Text Markup Language (HTML), Cascading Style Sheet (CSS), Document Object Model (DOM), sintaxe e recursos do ES6 (sexta versão da especificação do *javascript* endossada pela European Computer Manufacturers Association (ECMA)) , Node.js (um ambiente de execução *javascript* do lado do servidor) e o Node Package Manager (NPM) (GACKENHEIMER, 2015). Ele está posicionado na camada *view* da arquitetura Model-View-Controller (MVC) e uma das características mais importantes do React é o fato de que o programador pode criar componentes ou elementos HTML personalizados para construir interfaces mais eficientes.

A dinâmica de funcionamento de uma aplicação React passa pela compreensão do ciclo de vida de um componente React que nada mais é do que uma série de ações que acontecem ao longo da montagem e desmontagem do componente. Para melhor entendimento de como ocorrem tais comportamentos Santos (2018, p. 22) explica que à medida que o usuário interage com a aplicação, o estado do componente é modificado causando assim uma serie de atualizações no DOM e que para melhor identificar tais mudanças, o React estabeleceu uma série de eventos que são chamados à medida que as alterações vão ocorrendo. O Apêndice A demonstra como é realizado o ciclo de vida de um componente no React destacando os três momentos possíveis: *Mounting*, *Update*, *Unmount* (BISWAL,2019, p. 2; GACKENHEIMER,

2015). O React utiliza os conceitos de *state* (estratégia usada para armazenar dados locais de componentes antes de persisti-los em uma base de dados) e *props* (estratégia usada para passar parâmetros para componentes) para gerenciar os dados coletados e/ou exibidos na interface (GACKENHEIMER, 2015). Segundo Santos (2018, p. 21), os *states* são de extrema importância, sendo eles os responsáveis pelas mudanças durante a execução da aplicação e armazenam os valores atuais do componente causando as mudanças no ciclo de vida da aplicação.

Um dos aspectos a ser considerado em relação ao desenvolvimento em React refere-se à curva de aprendizagem da tecnologia. Neste quesito Santos (2017) afirma que o domínio em Node.js, módulos do NPM, configuração do WebPack e conhecimentos sobre JSX é uma forma de criar elementos para serem utilizadas como *templates* de aplicações React. Camargos et al. (2019) complementam que o conhecimento em ES6 ou ECMAScript 6 por parte do desenvolvedor é um passo importante que minimiza a curva de aprendizagem no React JS.

2.5 TRABALHOS CORRELATOS

São apresentados três trabalhos correlatos, sendo que suas características são parecidas com a ferramenta do presente artigo. No Quadro 1 deste estudo é detalhado o WINTERFELL, desenvolvido por Hathaway (2015) que descreve a ferramenta como um gerador de formulários complexos, com validações por meio de um esquema JSON. Ele foi desenvolvido especialmente para formulários de perguntas e respostas utilizando o tipo *wizard*. No Quadro 2 está descrita a ferramenta desenvolvida por Alles *et al*. (2015) com o nome de JSON-FORM, que tem como principal característica o suporte de *schema*, customização, validação dos componentes que compõem o formulário. No Quadro 3 é apresentado o editor HTML BLOCKLY EDITOR, desenvolvido por Coderz (2019) e tem como principal característica o arrastar e soltar que é utilizado de maneira a encaixar as peças para montar o formulário.

Quadro 1 – Winterfell

Referência	Hathaway (2015)
Objetivos	Ajudar no desenvolvimento de formulários web já incluindo validações dos campos
Principais funcionalidades	- Função de arrastar e soltar o componente para alinha no formulário. - Permite selecionar os elementos dos componentes. - Possibilita selecionar os atributos e modificá-los.
Ferramentas de desenvolvimento	- Utiliza React para renderizar o formulário. - JSON para validação e configuração do formulário.
Resultados e conclusões	Após testes da aplicação foi verificado que a ferramenta funciona adequadamente e executa as funções exibidas em tela corretamente. Sem uma opção de salvar o formulário, fica mais difícil de recuperar os dados caso seja necessário reutilizá-los ou alterá-los para uso posteriormente. A função de arrastar e soltar otimiza a construção dos componentes pelo usuário do formulário facilitando a personalização.

Fonte: elaborado pelo autor.

Quadro 2 - Json Forms

Referência	Ignacio *et al.* (2015).
Objetivos	Ajudar no desenvolvimento de formulários web com ênfase em habilitar outras partes deste utilizando condicionais que dependendo do conteúdo selecionado ou escrito em determinado elemento, o outro formulário dinamicamente é apresentado.
Principais funcionalidades	- Validação da dinâmica do formulário. - Adição de condicionais para mostrar outros elementos. - A escrita do JSON é a forma de criação dos elementos HTML.
Ferramentas de desenvolvimento	- HTML escrito com *framework bootstrap*. - JSON para validação com JSON SCHEMA.
Resultados e conclusões	É utilizado de forma mais técnica, pois é necessário que o usuário entenda das estruturas JSON para criar o formulário desejado. Com suas validações (JSON SCHEMA) e condicionais de exibição de elementos HTML na tela, a ferramenta se mostrou adequada principalmente pelo quesito segurança contra falhas de estrutura no formulário

Fonte: elaborado pelo autor.

Quadro 3 - Blockly Html Editor

Referência	Coderz (2019)
Objetivos	Facilitar no desenvolvimento de formulários Web utilizando conceito de blocos em encaixe para criar o formulário em sequência.
Principais funcionalidades	- Possibilidade de seleção dos elementos e arrastá-los diretamente para o bloco e encaixá-los. - Exibição da opção de *zoo-in* e *zoom-out* para demostrar todo o bloco do formulário montado. - Seleção de atributos do elemento ao clicar sobre ele.

	- Verificação em tempo real como o HTML está sendo montado em uma aba ao lado.
Ferramentas de desenvolvimento	- Blockly do Google.
Resultados e conclusões	Ferramenta muito fácil de usar e já é possível juntar os blocos e montar um Formulário sem demandar muitos conhecimentos de programação. Contudo, devido a não utilização de *frameworks* visuais como por exemplo *bootstrap*, após a geração o visual do formulário fica um pouco fora dos padrões atuais fazendo assim o programador implementar toda a parte visual e de estilos (CSS) dos elementos HTML.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DA PROTÓTIPO

Nesta seção são descritos os aspectos mais relevantes de especificação e implementação para a compreensão sobre o trabalho desenvolvido.

3.1 REQUISITOS DO TRABALHO

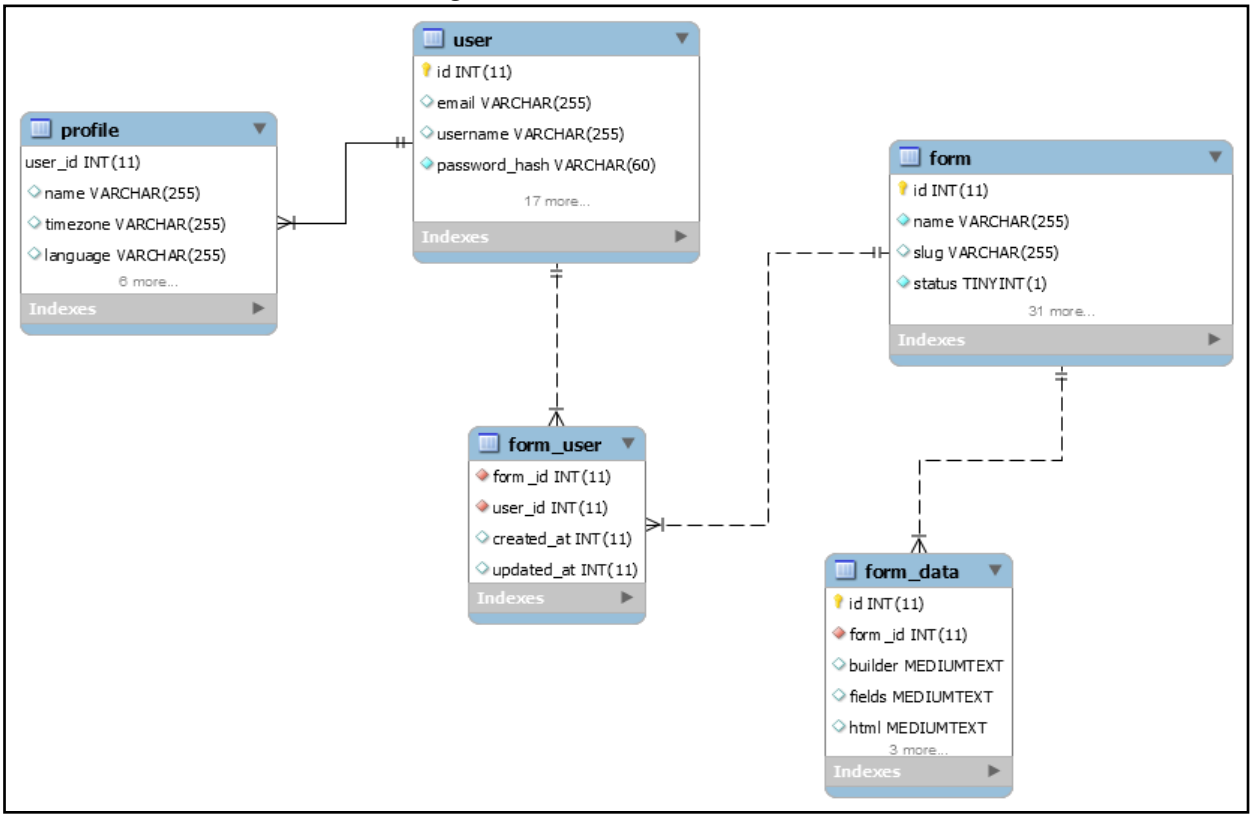
O protótipo desenvolvido neste trabalho deve apresentar os seguintes Requisitos Funcionais (RF) e não funcionais (RNF):

a) permitir ao usuário selecionar o elemento HTML que deseja utilizar (RF);
b) permitir ao usuário selecionar os atributos do elemento selecionado (RF);
c) permitir ao usuário arrastar o elemento para o formulário desejado (RF);
d) permitir ao usuário posicionar o elemento no formulário (RF);
e) permitir ao usuário salvar o JSON em um banco de dados (RF);
f) realizar conversão do JSON em código React (RNF);
g) disponibilizar ambiente em plataforma web (Requisito não funcional - RNF);
h) utilizar o banco de dados Mongo DB (RNF);
i) utilizar *framework bootstrap* para renderização (RNF);
j) utilizar a ferramenta WebForms para geração de formulários (RNF).

3.2 ESPECIFICAÇÃO

O projeto utiliza para construção dos formulários na web uma versão customizada da ferramenta Web Forms escrita em PHP. Esta ferramenta permite a edição e a execução de formulários na web de forma monolítica. A estratégia adotada envolveu a realização de adaptações no Web Forms de tal forma a permitir a geração de formulários em React similares aqueles construídos na própria ferramenta.

Figura 2 - Modelo de banco de dados



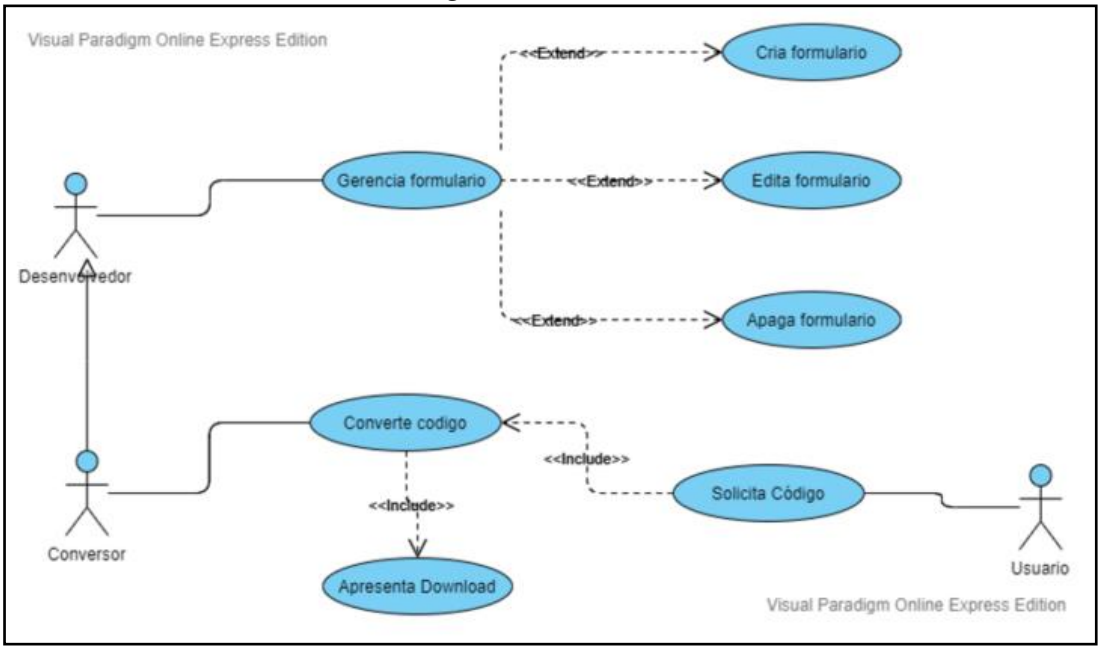
Fonte: Elaborado pelo autor.

A especificação do projeto utilizou alguns diagramas estruturais e comportamentais os quais são apresentados a seguir. A Figura 2 apresenta uma parte da estrutura de banco de dados utilizada pelo Web Forms, exibindo as entidades e os relacionamentos associados ao contexto do presente projeto, em que `user` seria então a tabela que guarda as informações do usuário do sistema com suas informações básicas e tem relação com a tabela de `form`, que é responsável por armazenas as informações básicas do formulário como nome, status entre outras. As tabelas `form` e `user` são ligadas através do relacionamento indireto de n para n que é a tabela `form_user`, que vai ligar o usuário ao formulário, e por último a tabela `form_data` que mantem as informações de customização do formulário e o JSON gerado.

3.2.1 Casos de uso

Nesta subseção são descritos os casos de uso da ferramenta utilizando Visual Paradigm Online. Na Figura 3 é possível identificar dois atores, denominados `Desenvolvedor` e `Conversor`. O ator `Conversor` tem acesso a funcionalidades que são indisponíveis para um desenvolvedor e o desenvolvedor é representado pelo ator `Desenvolvedor`.

Figura 3 - Casos de uso



Fonte: Elaborada pelo autor.

A seguir são detalhados os casos de uso:

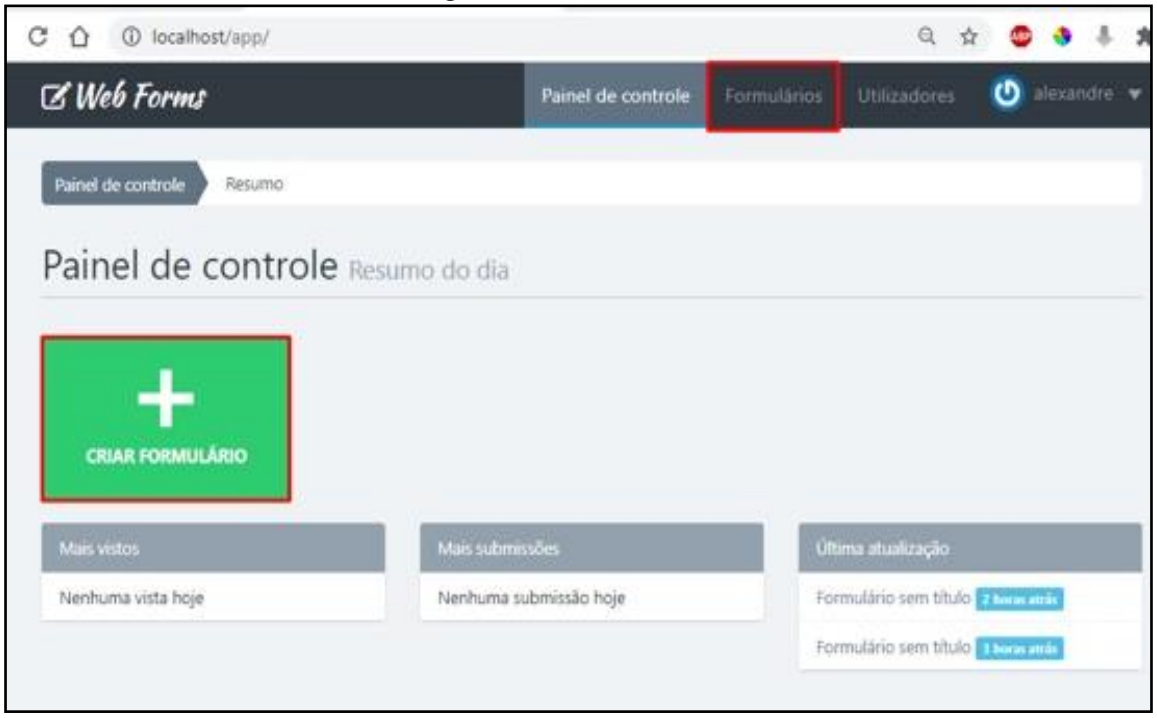
a) `Gerencia Formulário:` este caso de uso demonstra a relação entre o ator `Desenvolvedor` e a funcionalidade de `Gerência formulário.` Neste caso o `Desenvolvedor` possui duas opções para gerenciá-los, usando a tela de listagem de formulários criados e, após a efetivação da criação do formulário, o sistema direciona para a listagem onde é realizado o gerenciamento;
b) `Cria formulário`: é realizada a criação do formulário, onde são apresentadas as opções de elementos HTML a serem selecionados, configurações do formulário e local onde serão posicionados os elementos para a organização do formulário;
c) `Solicita Código:` é realizada a relação entre os atores `Conversor` e `Usuário,` o `usuário` seleciona a opção no gerenciador denominada `React` que será responsável por buscar os dados do formulário e entregar para o ator `Conversor` solicitar a conversão do código;
d) `Converte Código:` neste caso de uso é realizado então a busca dentro dos dados de cada elemento do formulário que foi enviado pelo ator `Conversor`, e após o término da conversão, é devolvido para o ator `Conversor` o código fonte convertido para `React`;
e) `Apresenta Download:` este caso de uso demonstra a interação entre o ator `Conversor` e o ator `Usuário`. O `Conversor` após receber a conversão do formulário executado pelo caso de uso `Converte Código` para código `React`, disponibiliza um arquivo no formato `.js` para uso posterior em uma aplicação `React`.

A ferramenta construída é composta de duas partes: um construtor de formulários na web e um módulo de conversão de modelos expressos em formato JSON para código fonte React. A próxima seção apresenta o funcionamento do gerador de formulários.

3.2.2 Gerador de formulários Web Forms

O gerador de formulários Web Forms tem como objetivo facilitar o desenvolvimento de formulários, apresentando uma interface que ajuda o desenvolvedor a criá-los de forma mais rápida. Inicialmente é preciso ter um usuário criado pelo administrador do sistema. Ao acessar a ferramenta é exibida a tela de *dashboard* (Figura 4) em que é possível visualizar os formulários criados ou iniciar a criação de um novo formulário clicando-se no botão `Formulários.`

Figura 4 - Tela de Dashboard

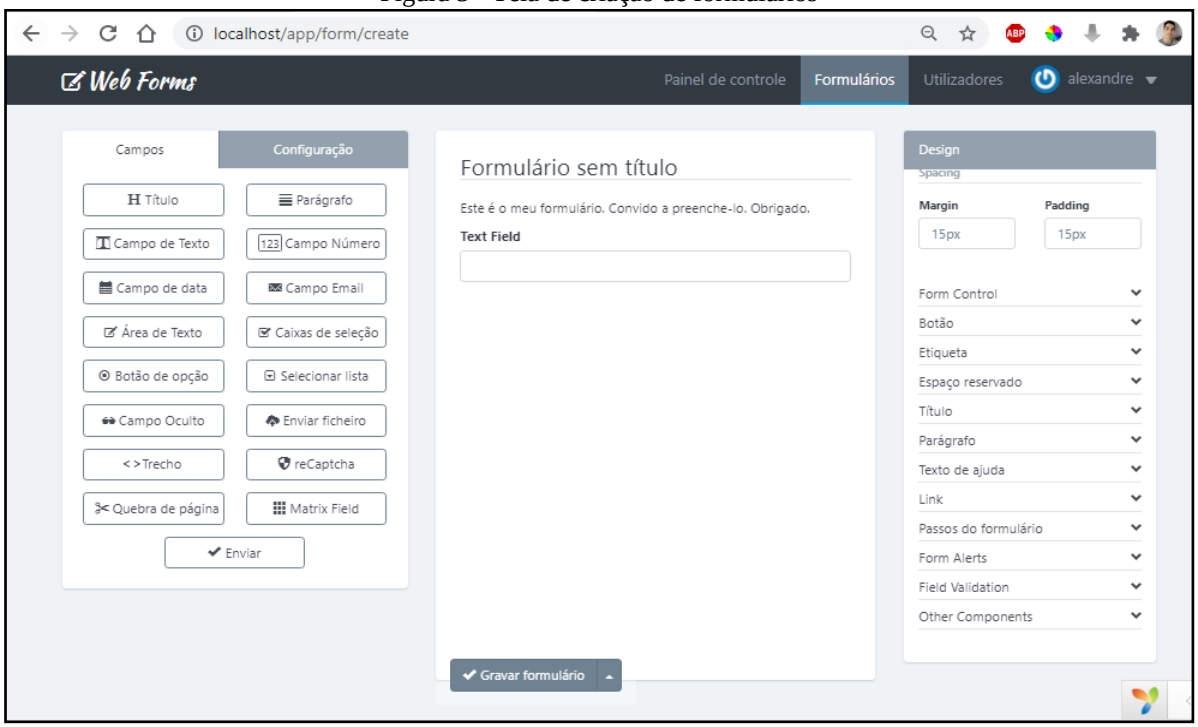


Fonte: Elaborado pelo autor.

A Figura 5 apresenta a tela com todas as opções principais disponíveis para a construção do formulário. A aba de `Campos` lista as opções de elementos para colocar no formulário e a aba `configurações` oferece ao usuário opções de identificação do formulário com um nome personalizado. A área central da tela é o espaço em que serão posicionados os componentes arrastados da aba `campos`. Finalmente a aba `Design` contém com os atributos gerais do formulário.

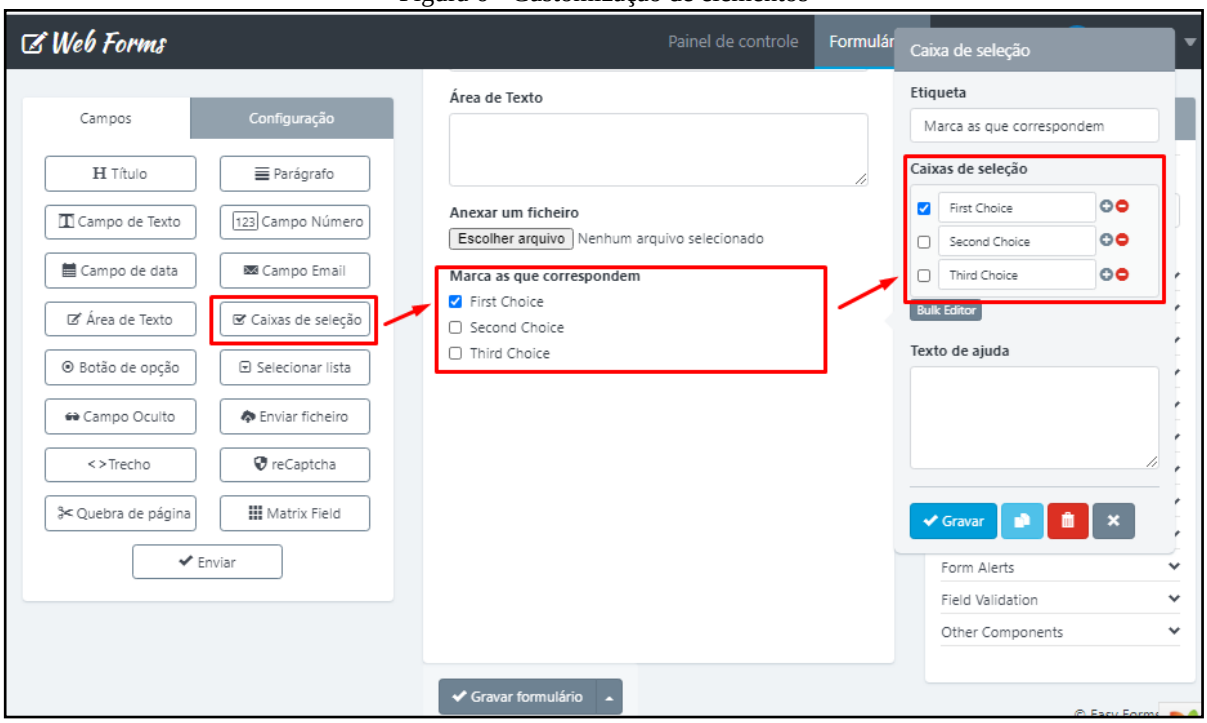
Para utilizar os elementos na aba `Campos` e posicioná-los no formulário, basta arrastar e soltar pois o elemento se posicionará automaticamente no espaço útil. Após posicionado, o próximo elemento selecionado pode ser colocado tanto na parte inferior do elemento anterior como na parte superior. Cada elemento possui características próprias, então para customizá-los é necessário um menu com as opções possíveis de customização dele. A Figura 6 destaca como exemplo o elemento Caixas de seleção. Ao selecioná-lo no formulário, será aberto um menu de customização para o elemento selecionado que é diferente para cada tipo de campo. Nesse caso tem-se a `Etiqueta`, um atributo apresentado como pergunta para seleção e Caixas de seleção, em que é possível criar as opções customizadas do campo.

Figura 5 - Tela de criação de formulários



Fonte: Elaborado pelo autor.

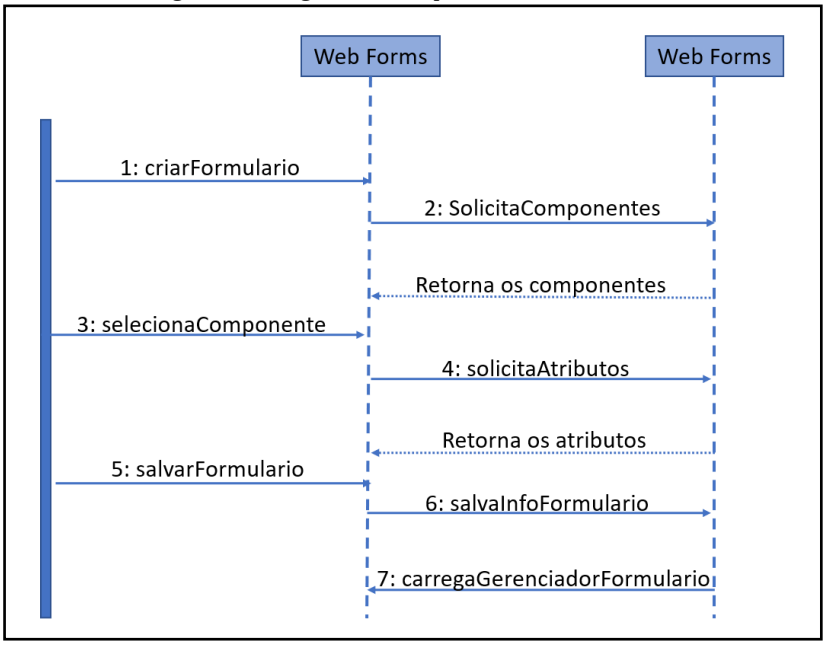
Figura 6 - Customização de elementos



Fonte: Elaborado pelo autor.

Depois de construído o formulário é salvo em formato JSON. O diagrama de sequência apresentado (Figura 7) tem como objetivo demonstrar o fluxo de sequência da funcionalidade de criação dos formulários.

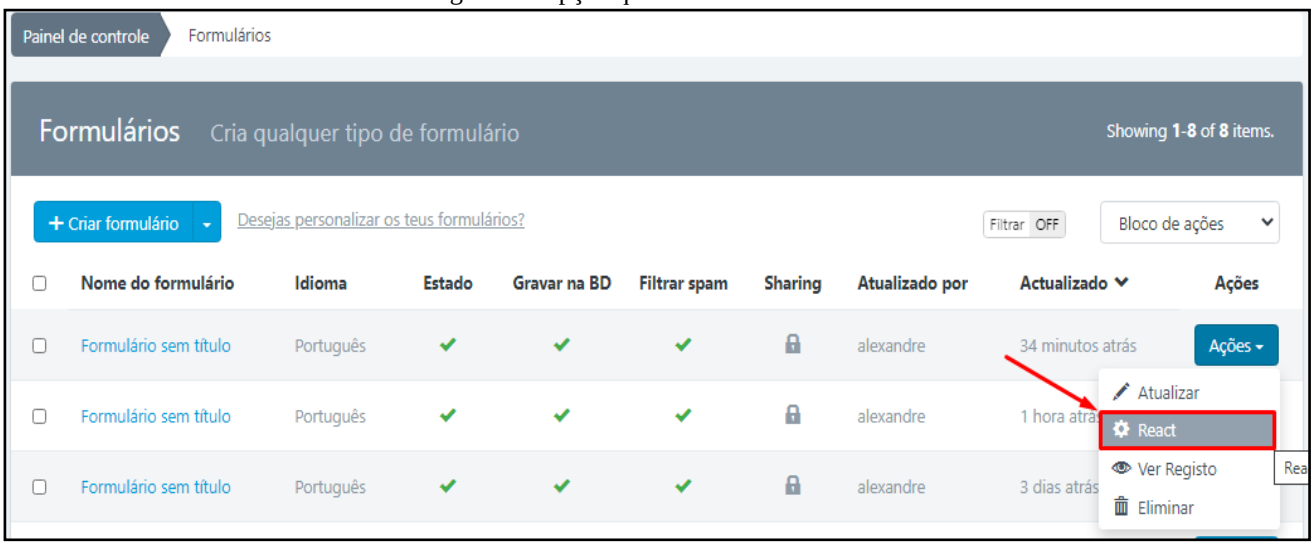
Figura 7 - Diagrama de sequência do Web Forms



Fonte: Elaborado pelo autor.

Após um formulário ter sido criado na ferramenta Web Forms é possível ativar a biblioteca externa (protótipo) para conversão e geração do código fonte React. Para viabilizar esta operação foi introduzido o botão `React` na lista de ações do formulário localizado na listagem de formulários criados no Web Forms (Figura 8).

Figura 8 - Opção que chama o conversor



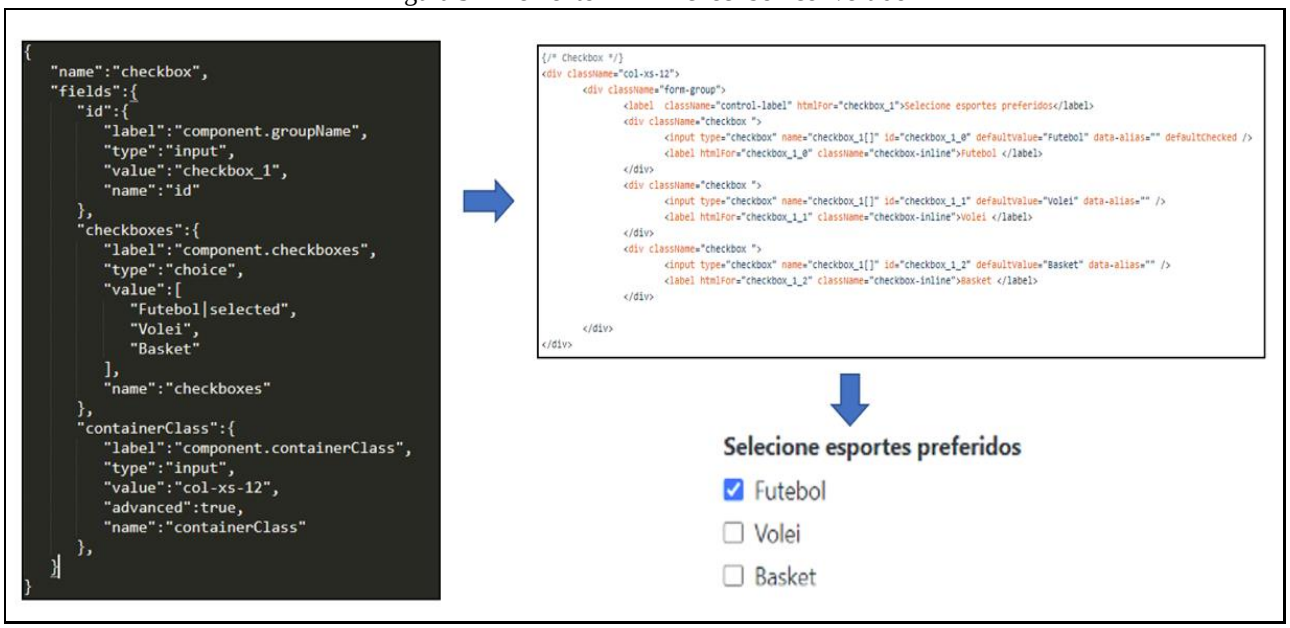
Fonte: Elaborado pelo autor.

Após clicar no botão `React` é ativado o código apresentado no Apêndice B (o qual foi desenvolvido justamente para viabilizar a transformação do modelo JSON) que recupera o modelo JSON da base de dados do Web Forms e produz um arquivo fonte contendo um componente React correspondente.

No Apêndice `C` é demonstrado um exemplo de como é capturado cada atributo do JSON de um componente `checkbox` e realizada a sua conversão para código fonte React.

O Figura 9 apresenta um trecho de código correspondente ao modelo JSON gerado, seu código HTML que é resultado da conversão e o resultado do HTML rodando em uma aplicação React.

Figura 9 - Elemento HTML checkbox convertido



Fonte: Elaborado pelo autor.

3.2.3 Ferramenta de conversão JSON para React

Conforme apresentado anteriormente, o protótipo desenvolvido parte de uma especificação de interface gerada em formato JSON (referido doravante no texto como 'modelo de origem'). Neste sentido, esta subseção descreve a estrutura do modelo de origem e a correspondente estrutura em React. O modelo de origem contempla o seguinte conjunto de componentes: `checkbox`, `heading`, `paragraph`, `text`, `number`, `email`, `textarea`, `pagebreak`, `radio`, `selectlist`, `date`, `file`, `button`, `recaptcha`, `matrix`, `snippet`. Um modelo de origem possui um conjunto de atributos básicos os quais são apresentados no Quadro 4, quais sejam: o nome do formulário a ser gerado e o formato de layout dele.

Quadro 4 - Configuração básica de um formulário expresso em formato JSON

```
{
  "settings":{
    "name":"Formulário sem título",
    "canvas":"#canvas",
    "layouts":[
      {
        "id":"",
        "name":"Vertical"
      }
    ],
  },
```

Fonte: Elaborado pelo autor.

Cada componente de interface que compõe o formulário é descrito dentro do atributo `initForm` no modelo de origem e este contempla todas as características particulares do componente. O Quadro 5 exibe um modelo de origem representando um formulário com um componente `heading` e os atributos `name`, `title` e `fields` que igual para todos os elementos. O `name` representa o identificador do elemento no JSON. O `title` representa o identificador do elemento no JSON, utilizado pelo Web Forms para controlar onde será exibido o título do elemento.

Quadro 5 - Atributo `initForm` no modelo de origem

```
"initForm":[
  {
    "name":"heading",
    "title":"heading.title",
    "fields":{
      "id":{
```

Fonte: Elaborado pelo autor.

A próxima subseção descreve as características do componente text e os demais componentes como *heading*, *checkbox*, *pagebreak*, *selectlist*, *recaptcha*, *snippet* e *matrix* são descritos no Apendice F.

3.2.3.1 Componente *Text*

Um componente *text* é utilizado para permitir a entrada de dados em campos texto. Possui as seguintes propriedades no modelo de origem as quais são apresentadas num exemplo no Quadro 6 (e seu respectivo formato de conversão para React). O atributo `fields` é um *array* que contém os seguintes atributos: `id`, `inputType`, `label` e `containerClass`. O `id` representa o identificador único do elemento. O `inputType` personaliza os tipos de variações que este campo possui (*text*, *color* e *password*). O elemento `label` contém o título e será exibido logo acima do elemento renderizado e `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.

Quadro 6 - Modelo de origem para componente text e respectivo código React

```
{
   "name":"text",
   "title":"text.title",
   "fields":{
      "id":{
         "label":"component.id",
         "type":"input",
         "value":"text_1",
         "name":"id"
      },
      "inputType":{
         "label":"component.inputType",
         "type":"select",
         "value":[
            {
               "value":"text",
               "selected":true,
               "label":"Text"
            },
              {
               "value":"color",
               "selected":false,
               "label":"Color"
            },
            {
               "value":"password",
               "selected":false,
               "label":"Password"
            }
         ],
      },
      "label":{
         "label":"component.label",
         "type":"input",
         "value":"Nome",
         "name":"label"
      },
        "labelClass":{
         "label":"component.labelClass",
         "type":"input",
         "value":"control-label",
         "advanced":true,
         "name":"labelClass"
      },
      "containerClass":{
         "label":"component.containerClass",
         "type":"input",
         "value":"col-xs-12",
         "name":"containerClass"
```

```
<!-- Text -->
<div class="col-xs-12">
    <div class="form-group">
        <label  class="control-label" for="text_1">Nome</label>
        <input type="color" id="text_1" name="text_1" value=""
         class="form-control"/>
    </div>
</div>
```

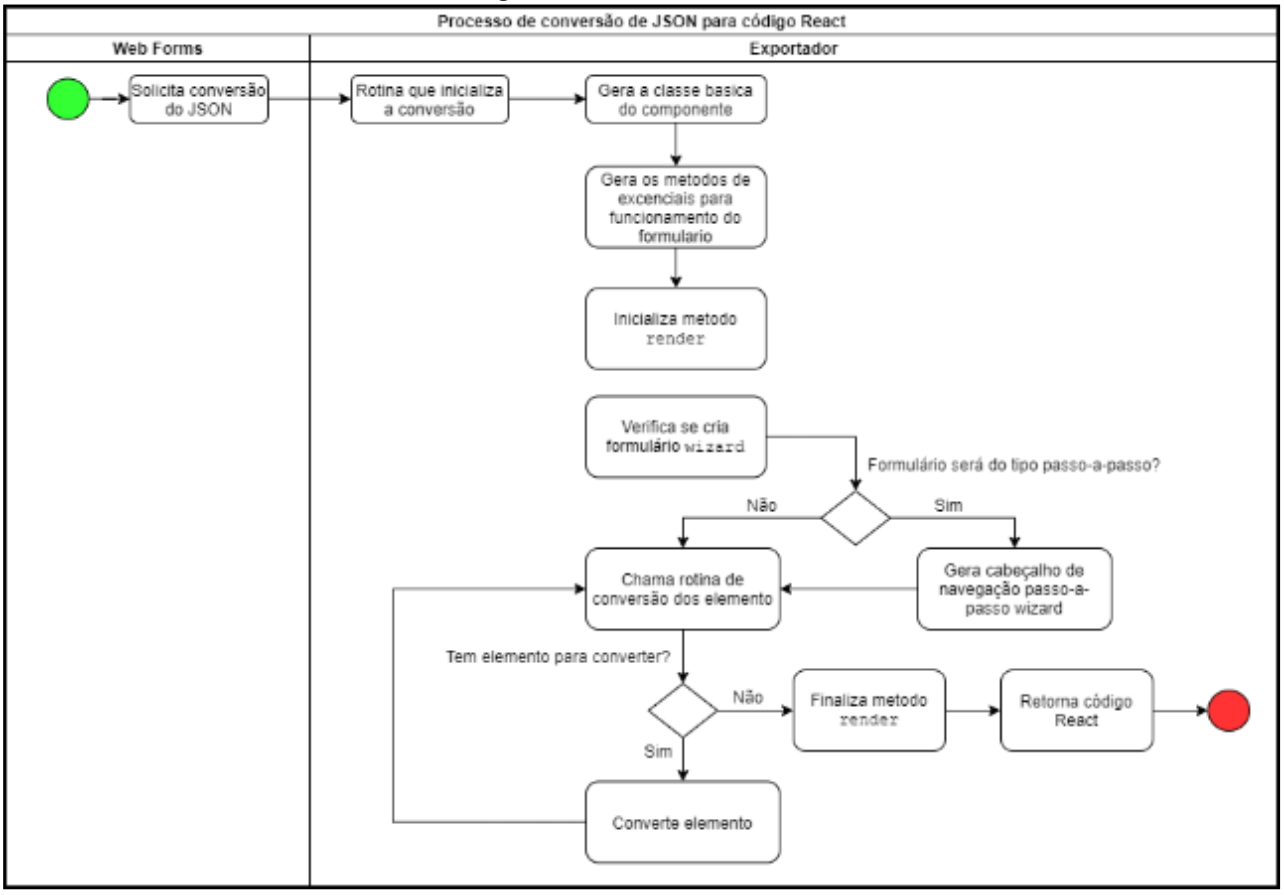
Fonte: Elaborado pelo autor.

Os componentes `number`, `email`, `textarea` e `date` não são explicados detalhadamente com os outros elementos, pois compartilham mesmos atributos fazendo deles similares ao componente `text`.

3.2.4 Diagrama de sequência do componente de conversão do modelo de origem em código React

O diagrama apresentado na Figura 10 tem como finalidade demostrar o fluxo para a solicitação da conversão do JSON em código React.

Figura 10 - Fluxo de conversão



Fonte: Elaborado pelo autor.

O processo de conversão do JSON para código fonte é realizado pela chamada do método `initInterpretador` que vai chamando métodos em cascata que são responsáveis pela conversão total do JSON para código fonte React. Cada elemento possui um método encarregado de realizar a tradução do formato em JSON para o formato HTML/React. Os métodos chamados em são:

a) `initReact`: É responsável por criar o componente React base e incluir os `imports` das bibliotecas do React, criação da classe do componente React, e em seguida chamar os métodos que criam as funções que compõem o ciclo de vida de um componente React;
b) `formReact`: é responsável por criar toda a estrutura do formulário dentro do método render do componente React, criando toda estrutura para o estilo *bootstrap* do formulário que é usado para deixar o formulário com mesmo padrão do formulário no Web Forms. Apêndice D demonstra o método principal do conversor;
c) `formStep`: responsável pela navegação do formulário, caso ele tenha sido criado no intuito de ser um formulário passo a passo do tipo *wizard*;
d) `elementCreate`: método que traduz cada elemento HTML do JSON convertendo-o para o elemento HTML React respectivo.

Cabe destacar que para que seja possível mostrar o formulário em React da mesma forma que é exibido no Web Forms, é necessário que sejam utilizadas bibliotecas de auxílio, como a biblioteca *bootstrap* para responsividade e navegabilidade, as bibliotecas CSS para estilização do formulário e ainda as de *javascript* para o funcionamento dos elementos utilizando funções. Como o WebForms não utiliza o recurso de NPM para gerenciar seus pacotes auxiliares, é necessário realizar a importação dos mesmos no arquivo principal HTML da aplicação React, o `index.html`. O Quadro 7 mostra como é realizada a importação das bibliotecas de CSS na `tag` `<head>` do HTML e o Quadro 8 exemplifica como é realizada a importação das bibliotecas auxiliares Javascript que ficam no final da `tag` `<body>` também no arquivo `index.html` na aplicação React.

Quadro 7 - Importação das bibliotecas CSS

```
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<!-- importação dosarquivos necessários de estilização do formulario -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/css/
bootstrap.min.css">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpreta
public.min.css">
```

Fonte: Elaborado pelo autor.

Quadro 8 - Importação das bibliotecas Javascript

```
  <!-- importação dos arquivos necessários de manipulação dos elementos do formulario -->
  <script src="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/js/libs/jquery.js" ></
  script>
  <script src="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/js/libs/
  signature_pad.umd.js" ></script>
  <script src="https://www.google.com/recaptcha/api.js"></script>

  <script src="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/js/libs/jquery.form.js" >
  </script>
  <script src="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/js/form.utils.min.js" ></
  script>
  <!-- <script src="https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@master/js/
  form.embed.min.js" ></script> -->
</body>
```

Fonte: Elaborado pelo autor.

Tendo em vista facilitar o uso da ferramenta de conversão construída, as bibliotecas auxiliares foram disponibilizadas em um CDN, permitindo que outras aplicações possam fazer uso dela desde que utilizem a mesma estrutura de modelo de origem JSON como descrito neste trabalho.

4 RESULTADOS

O Quadro 9 apresenta o comparativo entre o protótipo construído e os trabalhos correlatos. Analisando-se o quesito *Framework Front-End*, Hathaway (2015), e Alles et al. (2015), utilizam de *framework bootstrap* para estilizar os elementos e deixá-los visualmente bem dispostos no formulário, além de possuir a característica de responsividade. Codez (2019) já não alcança este quesito por gerar formulários sem estilo algum, deixando com uma aparência bem ultrapassada. No protótipo desenvolvido, o quesito foi atingido de forma plena, apresentando o formulário utilizando *bootstrap* para melhor apresentação visual.

Quadro 9 - Comparativo entre os correlatos

	Hathaway (2015)	Alles et al. (2015)	Coderz (2019)	Este trabalho
Framework Front-End	Sim	Sim	Não	Sim
Salvar o formulário	Não	Não	Sim	Sim
Interface Arrastar e soltar	Sim	Não	Sim	Sim
Seleção de elementos	Sim	Não	Sim	Sim
Seleção de atributos	Sim	Sim	Sim	Sim
Validação	Sim	Sim	Não	Não
Formulário *wizard*	Sim	Não	Não	Sim
Conversão para código React	Não	Não	Não	Sim

Fonte: Elaborado pelo autor.

No quesito *salvar o formulário*, Hathaway (2015) e Alles et al. (2015) não conseguiram atingir este objetivo, neste caso o usuário da ferramenta não pode recuperar o trabalho que já havia feito antes e consecutivamente não conseguira reaproveitar os formulários. Coderz (2019) e o protótipo desenvolvido conseguem tal feito, utilizando de gerenciamento de usuários. No quesito *Interface Arrastar e soltar*, apenas Alles et al. (2015) não conseguiu entregar uma interface amigável, nos demais trabalhos é possível construir um formulário apenas selecionando o elemento e posicionando no formulário como desejar. No quesito *Seleção de elementos*, novamente apenas Alles et al. (2015) não possui tal funcionalidade em função do formulário ser escrito todo em JSON e não possuir uma interface gráfica para a construção do formulário. No quesito *Seleção de atributos*, todos os correlatos e o presente trabalho possuem a funcionalidades, com ressalva para a ferramenta de Alles et al. (2015), pois ela permite que após gerar o formulário a partir do JSON, pode se manipular os atributos em modo visual e não mais somente pelo JSON. No quesito *Validação* tanto Hathaway (2015), quanto Alles et al. (2015) possuem tais validações, que tem como objetivo validar o formato e estrutura do JSON gerado e também as validações dos campos, por exemplo: quando não possui valor em um campo obrigatório, a aplicação apresenta uma mensagem de alerta. Neste quesito o protótipo não possui tal validação, a funcionalidade não pode ser implementada.

No quesito *Formulário Wizard*, Hathaway (2015) e o protótipo conseguem fazer de um formulário estático um formulário do tipo *wizard* que possibilita a divisão de um formulário em uma sequência de páginas contendo um agrupamento de campos com finalidade específica. Finalmente, no quesito *Conversão para código React* apenas o presente trabalho possui a funcionalidade de gerar código para componentes React a partir do JSON gerado.

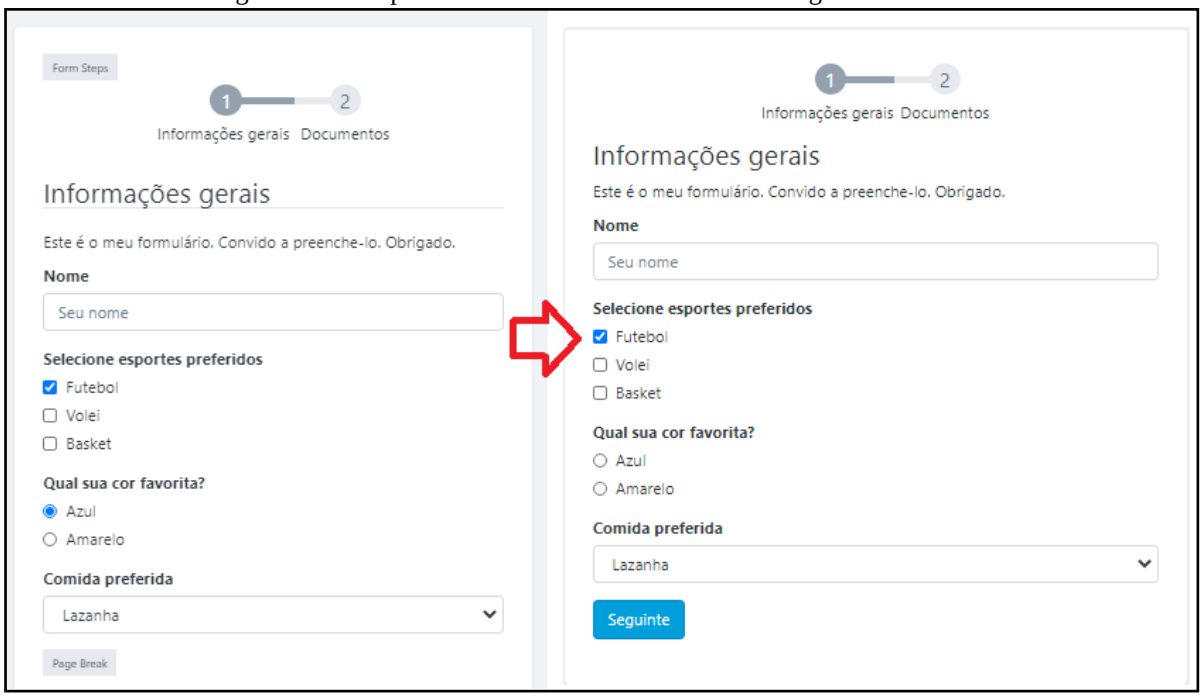
Como pode-se observar, apenas a funcionalidade de validação dos dados do formulário não foi alcançada em função do prazo de execução do projeto e a complexidade no processo de customização da ferramenta WebForms e a integração entre ela e o protótipo desenvolvido.

Em função do prazo exíguo, foi realizado um pequeno experimento com um especialista em desenvolvimento *frontend* de uma empresa de software da região de Blumenau. Para realizar o experimento foi disponibilizado um link externo para que o especialista pudesse conectar no computador de desenvolvimento e acessar a ferramenta. Após a realização do teste, este especialista indicou pontos em que a ferramenta se apresentou muito útil como a função de arrastar e soltar, que facilita na manipulação dos elementos no formulário, visual atraente, e facilidade na geração do componente ao término do processo de criação do formulário. O especialista parabenizou o trabalho desenvolvido fazendo uma ressalva que ele ficaria mais apropriado para ser utilizado em ambiente de produção caso visse a utilizar o modelo de gerenciamento de pacotes NPM ao invés da solução apresentada (por restrição do WebForms) de incluir as dependências dentro de um arquivo index.html.

A Figura 11 apresenta um formulário produzido pelo Web Forms (PHP) e a versão correspondente em React gerada pelo protótipo. Como se pode observar, a versão produzida guarda relação com a versão original. Comparando-se a solução construída em relação àquelas baseadas em formulários gerados em PHP (o Web Forms é um exemplo) é possível citar pelo menos as seguintes vantagens:

a) a solução em React possibilita a construção de web de página única mais complexas. Neste sentido, abre-se a possibilidade da migração de uma aplicação legada de uma forma mais automatizada contribuindo para minimizarem-se os custos desta operação;
b) quando uma página é alterada, o React atualiza apenas as partes da tela que foram alteradas tornando as atualizações de tela mais eficientes o que contribui, portanto, para uma melhor experiência de uso. Em PHP é necessário recarregar a página toda;
c) o React é mais flexível na construção de elementos em uma página e permite que os desenvolvedores criem código HTML que é configurado em tempo de programação com base em considerações de tempo de execução.

Figura 11 - Comparativo entre formulário WebForms e gerado em React



Fonte: Elaborado pelo autor.

5 CONCLUSÕES

O presente trabalho descreveu o desenvolvimento de um protótipo de uma ferramenta de criação de formulários Web e a conversão dos mesmos para código React. Ao longo do texto foram descritas as principais funcionalidades da

ferramenta e do conversor que tem como propósito facilitar na criação de um formulário Web permitindo reutilizá-lo em aplicações React.

A ferramenta Web Forms disponibiliza uma paleta de componentes que podem ser arrastados para um painel de composição do formulário. A partir desta composição o usuário seleciona a salvar o modelo em JSON e escolhendo a opção React a ferramenta automaticamente gera o código fonte React o qual pode ser embutido em uma aplicação Web tradicional. Apesar de não ter sido possível realizar uma bateria de testes mais robusta, buscou-se realizar uma demonstração do trabalho para um especialista em desenvolvimento Web. Embora o resultado do trabalho se caracterize como protótipo de prova de conceito, a avaliação deste especialista destacou aspectos positivos da iniciativa e pontuou alguns aspectos, os quais foram incluídos nas sugestões de futuras extensões como segue:

a) desenvolvimento da conversão também para outros *frameworks* como Angular, React Native;
b) criação de estilos e a importação do *bootstrap* pelo próprio React utilizando o NPM para gerenciamento dos pacotes utilizados na aplicação;
c) uso de uma biblioteca para gerenciar o formulário do próprio React, como React Hook Form entre outras;
d) utilização dos *react hooks* que permite gerenciar os componentes de forma melhorada;
e) conflitos na biblioteca do componente `Assinatura` e na biblioteca do React, não possibilitaram o funcionamento correto do elemento.

REFERÊNCIAS

BISWAL, Minati. React lifecycle methods. 2019, p. 2. Disponível em: https://www.researchgate.net/publication/337731350_React_lifecycle_methods. Acesso em: 06 dez. 2020.

CAMARGOS, João Gabriel Colares de; COELHO, José Flavio; VILLELA, Humberto Fernandes; ARAMUNI, João Paulo. Uma Análise Comparativa entre os Frameworks Javascript Angular e React. Computação e Sociedade, Belo Horizonte, v. 1, n. 1, p. 101-113, jul. 2019. Disponível em: http://www.fumec.br/revistas/computacaoesociedade/article/view/7307. Acesso em: 30 nov. 2020.

CODERZ. Blockly HTML Editor. 2020. Disponível em: https://coderz.ca/features/blockly-html-editor/, 2020. Acesso em: 18 out. 2020.

GAEA. **Veja como otimizar os processos de entrega de software**. 2020. Disponível em: https://gaea.com.br/veja-como-otimizar-os-processos-de-entrega-de-software/. Acesso em: 09 out. 2020.

GHOSH, Debasish. Learning to speak the language of the domain. *In*: _____. **DSLs in Action**. Stamford: Manning, 2011. p. 10-12. Disponível em: http://2.droppdf.com/files/mq4eW/dsls-in-action.pdf. Acesso em: 17 out. 2020

GONÇALVES, Debora. **Metodologias ágeis de desenvolvimento de software**: saiba mais sobre o assunto. 2019. Disponível em: https://blog.cronapp.io/metodologias-ageis-de-desenvolvimento-de-software/. Acesso em: 24 nov. 2020.

GUESSI, Milena; CRUZ, Wilmax.**Transformações modelo para texto**. 2012. Disponível em: https://edisciplinas.usp.br/pluginfile.php/123764/mod_resource/content/1/MDSE-book-slides-chapter9_2013.pdf. Acesso em: 08 nov. 2019.

HATHAWAY. **Winterfell**. 2015. Disponível em: https://github.com/andrewhathaway/Winterfell. Acesso em: 20 nov. 2015.

HERRINGTON, Jack. **Code generation in action**. Greenwich: Ct: Manning, 2003. 342 p.

ALLES, Ignacio del Valle et al. **Json-Forms**. 2015. Disponível em: https://github.com/brutusin/json-forms. Acesso em: 20 nov. 2020.

KLUG, Maicon. **Gerador de Código JSP Baseado em Projeto de Banco de Dados**. 2007. 18f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

KRILL, Paul. **Software engineers spend lots of time not building software**: Administrative tasks, brainstorming, and waiting for tests combine to overtake the hours spent designing and coding. 2013. Disponível em: https://www.infoworld.com/article/2613762/software-engineers-spend-lots-of-time-not-building-software.html. Acesso em: 21 out. 2020.

MASCARENHAS, Fabio. **Linguagens de Domínio Específico**. 2017. Disponível em: https://dcc.ufrj.br/~fabiom/dsl/. Acesso em: 04 set. 2020.

OLIVEIRA, Jessica Bassani. **Uma abordagem baseada em engenharia dirigida por modelos para suportar o teste de sistemas de software na plataforma de computação em nuvem**. 2012, 191 f. Dissertação (Mestrado) - Curso de Pós-Graduação em Egenharia, Universidade Federal do Maranhão, São Luis, 2012

RIVERO, José Matías et al. Mockup-driven development: providing agile support for model-driven web engineering. **Information and Software Technology,** Buenos Aires, v. 56, p. 670-687, 2011/2014.

ROSSI, Gustavo; URBIETA, Matias; DISTANTE, Damiano; RIVERO, Jose Matias; FIRMENICH, Sergio. **25 Years of model-driven web engineering. what we achieved, what is missing**. 2016. Clei Electronic Journal, v. 19, n. 3, p. 1. Disponível em: http://www.scielo.edu.uy/pdf/cleiej/v19n3/0717-5000-cleiej-19-03-00005.pdf. Acesso em: 04 nov. 2020.

SANTOS, Daniel.**Tabela de comparação entre Angular, React + Redux e Vue.js**. 2017. Disponível em: https://medium.com/@daniel.dia/comparação-entre-angular-react-redux-e-vue-js-a256d0fce8e0. Acesso em: 20 nov. 2020.

SANTOS, Leonardo Pires dos. **Desempenho de aplicações móveis utilizando implementação nativa ou frameworks multiplataformas**. 2018. 37 f. Trabalho de Conclusão de Curso (Engenharia da Computação) - Centro de Ciências Exatas e Naturais, Universidade Federal de Pernambuco, Recife.

SOUZA, Willian Oliveira. **Entendendo estado de componentes com React na prática**. 2018, p. 1. Disponível em: https://imasters.com.br/front-end/entendendo-estado-de-componentes-com-react-na-pratica. Acesso em: 06 dez. 2020.

TEECE, David; PETERAF, Margaret A.; HEATON, Sohvi**. Dynamic Capabilities and Organizational Agility:** Risk, Uncertainty and Entrepreneurial Management in the Innovation Economy. California, v. 58, n. 4, p. 13-35, Agosto, 2016. Disponivel em: https://escholarship.org/content/qt9w86b523/qt9w86b523_noSplash_b32a93004450e5cae3f6dc56e15e01fe.pdf. Acesso em: 15/11/2020.

VOELTER, Markus. **DSL Engineering**: Designing, Implementing and Using Domain-Specific Languages. Stuttgart: CreateSpace Independent Publishing Platform, 2013. 558 p. Disponível em: http://voelter.de/data/books/markusvoelter-dslengineering-1.0.pdf. Acesso em: 15 nov. 2020.

WHITTLE, Jon; HUTCHINSON, John; ROUNCEFIELD, Mark. **The State of Practice in Model-Driven Engineering**. 2014, p. 79. Disponível em: https://ieeexplore.ieee.org/document/6507223. Acesso em: 04 dez. 2020.

WHITTLE, Jon; HUTCHINSON, John; ROUNCEFIELD, Mark. The State of Practice in Model-Driven Engineering. **School of Computing and Communications**, Lancaster, v. 31, n. 3, p. 79-85, jan./abr. 2011. Disponível em: https://ieeexplore.ieee.org/document/6507223. Acesso em: 04 dez. 2020.

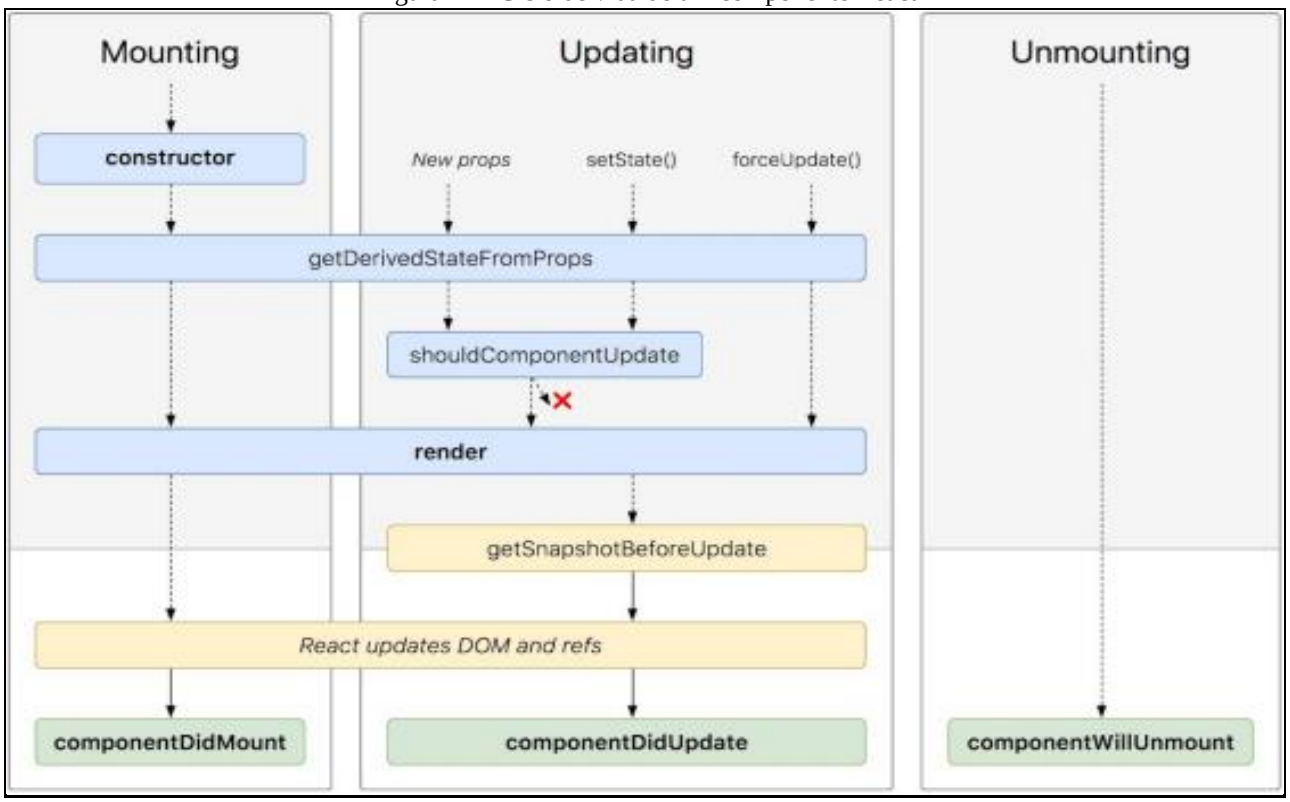
WUNDERLICH, Nico. **Exterminate ?:** who Influences IT Alignment and Digital Business Strategy**.** 2018 European Conference on Information Systems, ECIS (2018), p. 2. Disponível em: https://aisel.aisnet.org/cgi/viewcontent.cgi?artic

WUNDERLICH, Nico. Exterminate ?: who Influences IT Alignment and Digital Business Strategy. **European Conference on Information Systems**, Portsmouth, p. 2-3, nov, 2011. Disponível em: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1158&context=ecis2018_rp. Acesso em: 04 dez. 2020.

APÊNDICE A – CICLO DE VIDA DE UM COMPONENTE REACT

Neste apêndice é apresentado como funciona todo o ciclo de vida de um componente React conforme apresentado em Biswal (2019) (Figura 12).

Figura 12 - Ciclo de vida de um componente React



Fonte: Biswal (2019, p. 2).

APÊNDICE B – CHAMADA DA ROTINA DE CONVERSÂO PELO WEB FORMS

Este apêndice apresenta como é realizada a chamada da biblioteca do conversor, que é responsável por converter o JSON em código fonte React. É necessário buscar a biblioteca externa para ter acesso a rotina responsável pela conversão do Json para componente React demonstrado pela linha 566 do Quadro 10, e executar a rotina de conversão chamando o método `initInterpretador` da biblioteca, que retornará o código convertido e é exemplificado na linha 542 do Quadro 10. A linha 546 disponibiliza o download do arquivo com extensão *.js*.

Quadro 10 - Chamada do Conversor pelo Web Forms

```
529  $js = <<< SCRIPT
530
531  $(function () {
532      // Tooltips
533      $("[data-toggle='tooltip']").tooltip();
534
535      $(".react").click(function(){
536
537          $.ajax({
538              url: $(this).data("url"),
539              type: 'get',
540              success: function(data) {
541                  json_data = JSON.parse(data)
542                  let reactcode = initInterpretador('react', json_data);
543                  console.log(reactcode)
544                  var element = document.createElement('a');
545                  element.setAttribute('href', 'data:text/plain;charset=utf-8,' + encodeURIComponent(reactcode));
546                  element.setAttribute('download', 'formulario.js');
547
548                  element.style.display = 'none';
549                  document.body.appendChild(element);
550
551                  element.click();
552
553                  document.body.removeChild(element);
554
555              },
556              error: function (request, status, error) {
557                  alert(request.responseText);
558              }
559          });
560      })
561  });
562
563  SCRIPT;
564  // Register tooltip/popover initialization javascript
565  $this->registerJs($js);
566  $this->registerJsFile(
567      'https://cdn.jsdelivr.net/gh/romeroleonardoalexandre/interpretador_json_html@interpretador/index.js',
568      //'@web/static_files/js/interpretador.js',
569      ['depends' => [\yii\web\JqueryAsset::className()]]
570  );
```

Rotina da biblioteca que realiza a conversão

Busca a biblioteca de conversão

Fonte: Elaborado pelo autor.

APÊNDICE C – METODO DE CONVERSÃO DE UM ELEMENTO CHECKBOX

Este apêndice demonstra como é realizada o processo de criação de um elemento *checkbox* a partir da especificação JSON. No Quadro 11, é demostrado detalhadamente como é realizado a conversão. Na linha 344 é inicializado uma variável chamada `loop` que guardará o resultado de toda conversão dos valores a serem selecionados pelo usuário, e na linha 348 é chamada uma função do React que é responsável pela alteração dos valores do elemento quando o componente for utilizado em uma aplicação React.

Após a conversão dos valores, a linha 361 mostra como a variável `loop` será ainda incorporada a um agrupador HTML (linha 369) que retornará com o elemento *checkbox* convertido.

Quadro 11 - Código da geração de um elemento checkbox em React

```
341
342         //Metodo que cria o input type checkbox
343          checkbox (data)  {
344                 let loop = ""
345                 for (var i = 0; i < data.fields.checkboxes.value.length; i++) {
346                         let split = data.fields.checkboxes.value[i].split("|")
347                         loop += `<div className="checkbox ">
348                                 <input type="checkbox" onChange={(event)=>this.handleChange(event)}
349                                         name="${data.fields.id.value}[]"
350                                         id="${data.fields.id.value}_${data.fields.checkboxes.value[i]}"
351                                         defaultValue="${data.fields.checkboxes.value[i]}"
352                                         data-alias="" ${split.length > 1 ? "defaultChecked" : ""} />
353                                 <label htmlFor="${data.fields.id.value}_${data.fields.checkboxes.value[i]}"
354                                         className="checkbox-inline">
355                                         ${split[0]}
356                                 </label>
357                         </div>
358                         `
359                 }
360
361                 return `
362                 <div className="${data.fields.containerClass.value}">
363                         <div className="form-group">
364                                 <label className="${data.fields.labelClass.value}"
365                                         htmlFor="${data.fields.id.value}">
366                                         ${data.fields.label.value}
367                                 </label>
368
369                                 ${loop}
370                                 <span id="${data.fields.id.value}"></span>
371                         </div>
372                 </div>
373
374                 `
```

Fonte: Elaborado pelo autor.

APÊNDICE D – METODO DE CONVERSÃO DE UM ELEMENTO CHECKBOX

Neste apêndice é demonstrado o processo de criação do formulário e a conversão de cada elemento. O Quadro 12 explica como é realizada a conversão. O método `formReact` na linha 549 é responsável pela montagem de todo o formulário e cria os elementos do formulário como *input*, *checkbox*. E verifica o tipo de formulário, se é passo-a-passo (Wizard) e se precisa de uma barra de navegação. A partir da linha 551 até a linha 575 é realizado a conversão do JSON com estrutura do formulário para código React e utilizando Boostrap para lidar com o visual.

Na linha 561 e 562 o metodo `formSteps` verifica se há a necessidade de criar o formulário do tipo passo-a-passo (Wizard), é então verificado se existe um elemento de nome `formStep` dentro da estrutura do JSON para realizar a conversão. Após é chamado o metodo `elementCreate` como destacado na linha 564, é responsável por ler cada elemento dentro do JSON e realizar a converter para um elemento HTML React, o processo é realizado de forma sequencial, lendo cada elemento no *array* dentro do JSON. Após a conversão dos elementos, é chamado o metodo `progressBar` que verifica se há a necessidade de criar uma barra de progresso, que demonstrado nas linhas 567 e 568.

Quadro 12 - Método principal da criação do formulário

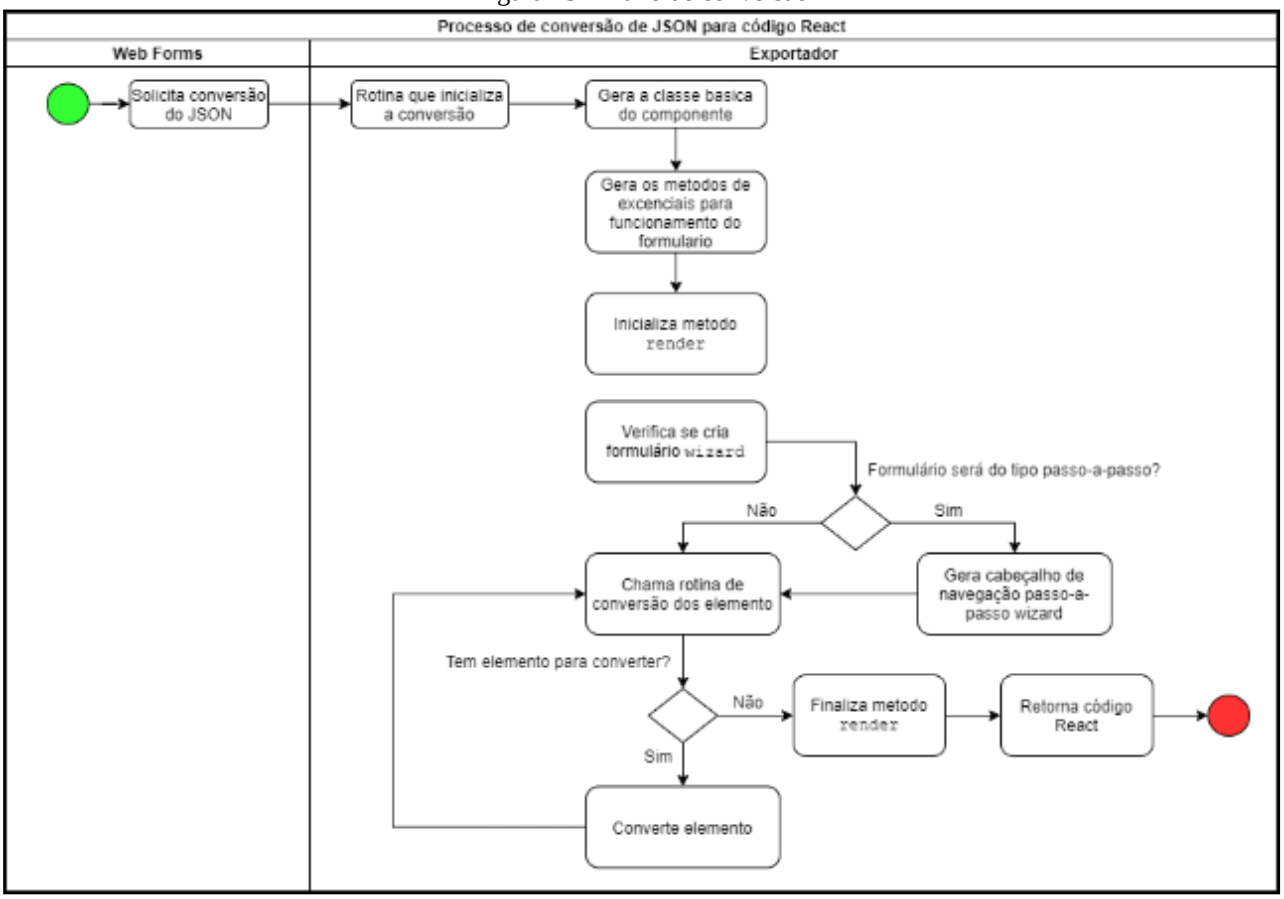
```
549  formReact (form_data)  {
550          return `
551          <div className="container">
552                  <div className="row">
553                          <div className="col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
554                                  <div className="form-view">
555                                          <div className="panel panel-default">
556                                                  <div className="panel-body">
557                                                          <div className="form-container">
558                                                                  <div id="messages"></div>
559                                                                  <form action="#" method="post" encType="multipart/form-data"
560                                                                          id="form-app">
561                                                                          ${typeof form_data.settings.formSteps !== 'undefined' ?
562                                                                                  this.formSteps(form_data.settings.formSteps) : ""}
563                                                                          <fieldset className="row" data-index="0">
564                                                                          ${this.elementCreate(form_data.initForm)}
565                                                                          </fieldset>
566                                                                  </form>
567                                                                  ${typeof form_data.settings.formSteps.progressBar !== 'undefined' ?
568                                                                          this.progressBar() : ""}
569                                                          </div>
570                                                  </div>
571                                          </div>
572                                  </div>
573                          </div>
574                  </div>
575          </div>
576
577          `
578  }
```

Fonte: Elaborado pelo autor.

APÊNDICE E – FLUXOGRAMA DE ATIVIDADE DO PROCESSO DE CONVERSÃO

Neste apêndice é demonstrado todo o processo de conversão do JSON em HTML/React. A Figura 13 explica como é realizada a conversão.

Figura 13 - Fluxo de conversão



Fonte: Elaborado pelo autor.

APÊNDICE F – CONVERSÂO DOS DEMAIS ELEMENTOS DO JSON PARA COMPONENTE REACT

Um componente *heading* é utilizado para inserir rótulos de títulos nos padrões HTML (h1, h2, h3, h4, h5 e h6). Possui a propriedade `fields` que é um *array* que contém os seguintes atributos: `id`, `text`, `type` e `containerClass`. O `id` define o identificador único do elemento. O `text` define o texto de exibição do componente e o elemento `type` contém as características como *H1, H2*, etc. O atributo `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.

O componente *checkbox* tem por característica a seleção de várias opções e possui a propriedade `fields` que contém os seguintes atributos: `id, label`, `checkboxes` e `containerClass`. O `id` representa o identificador único do elemento. O `label` define o texto de exibição do componente. O elemento `checkboxes` contém o conjunto de opções que serão apresentadas para seleção. Já `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente. Um componente *radio* é similar ao componente `checkbox`, mas somente uma das opções pode ser selecionada.

Um componente *pagebreak* é utilizado para definir os locais onde serão inseridas as *tags* de controle de paginação e possui a propriedade `fields` que contém os seguintes atributos: `id, prev`, `next`. O `id` representa o identificador único do elemento. O `prev` determina se o elemento apresentará o botão `Anterior`. O elemento `next` determina se o elemento vai possuir o botão `Seguinte`.

Componente *selectlist* representa um controle que apresenta um menu de opções. Cada opção dentro do menu é representada por um elemento *tag option* e as opções podem ser pré-selecionadas para o usuário. Possui a propriedade `fields` que contém os seguintes atributos: `id, label`, `options` e `containerClass`. O `id` representa o identificador único do elemento. O `label` define o texto de exibição do componente. O elemento `options` contém o conjunto de opções que serão apresentadas para seleção. Já `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.

Um componente *recaptcha* representa uma caixa de diálogo que apresenta desafios ao mesmo para proteger áreas restritas de sites de acesso por robôs. Possui propriedades como `fields` que contém os seguintes atributos: `id, theme` e `containerClass`. O `id` representa o identificador único do elemento. O `theme` define se o tema de exibição do componente será *light* ou *dark*. Já `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.

Componente do tipo *snippet* representa uma pequena seção de texto ou código fonte que pode ser inserido no código de um programa ou página web e tem o propósito de facilitar a implementação de trechos e/ou funções comumente utilizados em uma seção maior do código. Também possui a propriedade `fields` que contém os seguintes atributos: `id, snippets` e `containerClass`. O `id` representa o identificador único do elemento. O `snippets` armazena o trecho de código para exibição. Já `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.

Um componente *matrix* permite armazenar as características de um campo do tipo tabela o qual pode conter várias linhas que podem ser perguntas e várias colunas com possíveis respostas sendo utilizado um recurso do tipo *checkbox* para marcação das alternativas. Possui propriedades como `fields` que é um *array* que contém os seguintes atributos: `id, inputType,label,questions,answers` e `containerClass`. O `id` representa o identificador único do elemento. O `inputType` representa o tipo do elemento que será inserido dentro da tabela e receberá os valores inseridos pelo usuário, neste caso o `checkbox`. O `label` define o título de exibição do elemento. O `questions` representa a pergunta que será feita em cada linha da tabela. O `answers` representa as possíveis respostas para cada linha da tabela. Já `containerClass` contém os elementos que descrevem características da classe de CSS utilizada para renderizar o componente.