UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTÁTISTICA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Thiago Miklan Moreira

# BIBLIOTECA DE COMPONENTES REACT INSPIRADOS NO FRAMEWORK CSS BULMA

Florianópolis, novembro de 2020
Thiago Miklan Moreira

# Biblioteca de componentes React inspirados no framework CSS Bulma

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Sistemas de Informação

Orientador: Prof. Leandro José Komosinski, Dr.

Florianópolis

2020

Thiago Miklan Moreira

**Biblioteca de componentes React inspirados no framework CSS Bulma**

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Curso Sistemas de Informação

Florianópolis, novembro de 2020.

_____
Prof. Cristian Koliver, Dr.
Coordenador do Curso

**Banca Examinadora:**

_____
Prof. Leandro José Komosinski, Dr.
Orientador
Universidade Federal de Santa Catarina

_____
Prof. Jean Carlo Rossa Hauck, Dr.
Avaliador
Universidade Federal de Santa Catarina

_____
Prof. Alex Sandro Roschildt Pinto, Dr.
Avaliador
Universidade Federal de Santa Catarina

## SUMÁRIO

## AGRADECIMENTOS

Agradeço a Deus pela ajuda nesses difíceis anos de curso, agradeço a todos da minha família que de alguma forma me ajudaram (citar todos seria muito difícil), a minha namorada pela força e paciência comigo e a todos que torceram ou contribuíram durante esse árduo processo. Agradeço também ao professor Leandro Komosinski pela ajuda ao longo de toda a orientação do TCC e pela sua disposição na orientação do trabalho. Um agradecimento especial para minha mãe, Marilene, pelos anos de trabalho, luta, disposição, resistência, força e apoio em todos momentos da minha vida, nas fases felizes e nas fases mais sombrias.

Muito obrigado a todos.

## RESUMO

No desenvolvimento Web, as linguagens CSS, HTML e JavaScript predominam no mercado como soluções para a criação de sistemas web dos mais variados portes, com a crescente adesão do JavaScript por parte dos desenvolvedores, diversos frameworks e bibliotecas foram criados para darem auxílio ao desenvolvimento de sistemas com essas tecnologias, ao observar que a integração das três linguagens para criação de um sistema web acarreta em uma quantidade considerável de repetição de código. O React é uma biblioteca JavaScript para construção de interfaces gráficas, que busca oferecer aos desenvolvedores que utilizam JavaScript um reaproveitamento dos códigos criados para a construção das interfaces. O Bulma é um framework CSS que predefine diversos elementos que podem ser utilizados pelos desenvolvedores JavaScript para manipular as aparências das aplicações desenvolvidas. Ao usar o Bulma em conjunto com o React é introduzido um problema, a grande quantidade de código HTML repetido na integração das duas tecnologias, contrapondo a ideia primária dos frameworks e bibliotecas de facilitar o reuso de software, com o intuito de melhorar o reuso de software ao utilizar o React e o Bulma em conjunto, desenvolveu-se no presente trabalho uma biblioteca *open source* de componentes React implementados especificamente para utilização com o framework CSS Bulma. Ao finalizar o presente trabalho, constatou-se que é possível diminuir consideravelmente o código HTML necessário fornecendo melhor produtividade ao utilizar React e Bulma.

**Palavras-chave:** React. Bulma. Componentes.

## ABSTRACT

In Web development, CSS, HTML and JavaScript predominate in the market as solutions for the creation of web systems of the most varied sizes, with the growing adhesion of JavaScript by developers, several frameworks and libraries were created to assist the development of systems with these technologies, when observing that the integration of the three languages to create a web system causes a considerable amount of code repetition. React is a JavaScript library for building graphical interfaces, which seeks to offer developers who use JavaScript a reuse of the codes created for the construction of interfaces. Bulma is a CSS framework that predefines several elements that can be used by JavaScript developers to manipulate the appearance of developed applications. When using Bulma in conjunction with React, a problem was introduced, the large amount of HTML code repeated in the integration of the two technologies, opposing the primary idea of frameworks and libraries to facilitate software reuse. In order to improve software reuse when using React and Bulma together, a library of React components developed specifically for use with the CSS Bulma framework was developed in the present work. At the end of this work, check if it is possible to considerably decrease the HTML code needed to improve the use of React and Bulma.

**Keywords:** React. Bulma. Components.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

JS JavaScript

JSX JavaScript XML

DOM Document Object Model

CSS Cascading Style Sheets

HTML HyperText Markup Language

ARB Assemble React Bulma

BLE Brightleaf Elements

RBC React Bulma Components

NPM Node Package Manager

UFSC Universidade Federal de Santa Catarina

# 1 INTRODUÇÃO

As aplicações web são sistemas que operam em navegadores na Internet e são construídos a partir de várias linguagens de programação. Os desenvolvedores de aplicações web, na maioria dos casos, necessitam utilizar pelo menos três linguagens: JavaScript, HTML e CSS.

O uso do JavaScript aumentou consideravelmente nos últimos anos. O JavaScript foi a linguagem com mais repositórios criados na plataforma de versionamento de código GitHub desde o ano de 2014, sendo a linguagem mais utilizada pelos colaboradores da plataforma. No site StackOverFlow, mostra que em 2020 as três mais populares tecnologias do site foram JavaScript, HTML e CSS (na categoria linguagens de marcação, de script ou de programação) (STACKOVERFLOW,2020; GITHUB,2019).

Com o crescimento acentuado dos desenvolvedores de aplicações web, por efeito colateral, desenvolveu-se uma quantidade considerável de frameworks e bibliotecas, que visam facilitar o trabalho dos programadores através principalmente do reuso de software.

O React é uma biblioteca JavaScript que busca auxiliar na criação de interfaces gráficas de usuário. O React é baseado em componentes e possui uma característica de dinamismo na exibição das interfaces, onde cada componente é renderizado conforme o estado da aplicação (REACT,2020).

O Bulma é um *framework* CSS que oferece uma padronização da aparência das páginas Web, permitindo que os programadores consigam criar facilmente uma interface gráfica com um nível aceitável de estética (BULMA,2020). Optou-se pelo Bulma como framework CSS pela sua simplicidade, responsividade e estética, além da confiança de que o framework pode ser mais utilizado pelos desenvolvedores e possuir qualidades que se destacam frente a outros frameworks.

É comum no desenvolvimento web a integração de frameworks e bibliotecas, o React e o Bulma podem e são usados em conjunto para criação de interfaces gráficas. O principal problema que surge com a integração dessas tecnologias é a grande repetição de código HTML.

Observando a grande quantidade de código HTML produzido pela integração entre React e Bulma, a falta de trabalhos construídos especificamente para essa

integração e pela importância que o reuso de software possui no desenvolvimento de aplicações, o presente trabalho busca oferecer um conjunto de componentes desenvolvidos especificamente para a integração Bulma/React visando melhorar o reuso de software ao se utilizar as duas tecnologias, diminuindo o código HTML necessário para esse uso.

## 1.1 OBJETIVOS

Nas seções 1.1.1 e 1.1.2 são descritos os objetivos do presente trabalho.

### 1.1.1 Objetivos Principais

Desenvolver um conjunto de componentes React inspirados no framework CSS Bulma que auxiliarão no desenvolvimento de aplicações que utilizem React e Bulma.

### 1.1.2 Objetivos Específicos

- Redução de aproximadamente 25% das tags HTML necessárias após aplicação do presente trabalho (refactoring).

### 1.1.3 Metodologia

A metodologia do trabalho pode ser dividida em 4 etapas, que são:

- Etapa de análise
- Etapa de aprendizado teórico e prático
- Etapa de desenvolvimento
- Etapa de publicação

#### *1.1.3.1 Etapa de análise*

Essa etapa correspondeu a exploração inicial dos assuntos tratados no trabalho, contando com o uso básico das tecnologias e uma informal pesquisa teórica,

buscando sempre conhecer um pouco de cada assunto necessário para a construção do trabalho, em outras palavras, uma fase de adaptação e início.

#### *1.1.3.2 Etapa de aprendizado teórico e prático*

Para que a construção da solução fosse realizada de maneira correta, explorou-se os conceitos e práticas necessárias, através da pesquisa teórica e tecnológica (revisão bibliográfica) e utilização prática das tecnologias.

É necessário detalhar "pesquisa teórica e tecnológica", pois na fundamentação do presente trabalho existem os conceitos teóricos como componentes, programação funcional e framework, porém existem também os conceitos tecnológicos como Javascript, Flow e Bulma por exemplo. A principal diferenças entre os dois tipos é que a parte teórica trata de conceitos e a parte tecnológica trata de ferramentas, linguagens de programação, bibliotecas e frameworks específicos.

Na revisão bibliográfica, inicialmente, diversos tópicos essenciais para o entendimento do trabalho foram selecionados. A partir de um primeiro levantamento desses tópicos, iniciou-se uma pesquisa teórica para fundamentar esses conceitos, sendo essa pesquisa realizada com livros, artigos científicos, artigos de blogs, trabalhos de conclusão de curso, jornais, teses e outros. Após o levantamento dos tópicos iniciais e pesquisa do material necessário, aconteceu um refinamento ao decorrer da pesquisa, excluindo-se itens desnecessários, adicionando itens importantes e mantendo os essenciais para a compreensão do trabalho (como componentes, React e outros).

O aprendizado prático das tecnologias envolvidas foi realizado de três formas, através da criação de pequenas aplicações para explorar essas tecnologias, através da reprodução de exemplos fornecidos nas documentações das tecnologias e através da análise de códigos encontrados em sites, blogs e no GitHub.

#### *1.1.3.3 Etapa de desenvolvimento*

Após a etapa inicial de aprendizado teórico e prático, iniciou-se o desenvolvimento dos componentes que fazem parte da solução.

Para delimitar o escopo, utilizou-se a documentação fornecida pelo Bulma, onde existem diversos elementos que o Bulma disponibilizada para uso. Para cada elemento existente no Bulma foi analisada a sua importância e utilidade e decidido se integraria ou não a solução. Após escolher um elemento Bulma o seu respectivo componente entra em desenvolvimento, sendo analisada a sua documentação no Bulma, realizado testes práticos, analisado o seu modelo ("esqueleto do componente" e então codificado.

Após a codificação, realizou-se um teste unitário para cada exemplo fornecido na documentação do Bulma e efetuava-se também outros testes julgados necessários. Após os testes unitários, elaborou-se testes na ferramenta Jest, a qual permitia encontrar atualizações indesejadas ao longo do desenvolvimento da solução.

Realizando essa primeira parte de testes, o componente é aplicado em um pequeno sistema para verificar se sua utilização apresentava erros ou dificuldades exacerbadas de uso.

Observa-se que alguns componentes surgiram da aplicação prática da solução e da observação de necessidades não encontradas apenas com o componente sendo testado isoladamente. Pontua-se também que muitas melhorias foram realizadas ao se testar os componentes incorporados a uma solução.

Após atingir um montante considerável de componentes, foi gerada uma versão de desenvolvimento no NPM e desenvolvida a documentação inicial da solução.

Existindo uma versão minimamente estável da solução, aplicou-se o *refactoring* de pequenas aplicações construídas apenas com React e Bulma (sem nenhuma solução auxiliar), buscando entender a redução de código HTML gerada pela solução. No *refactoring*, foram definidas três métricas: porcentagem de redução do código HTML, porcentagem de redução do código total e porcentagem de redução do número de tags. No *refactoring*, foram selecionados componentes que faziam sentido (componentes grandes principalmente) e aplicou-se a solução para a criação do mesmo componente. Na seção 5 são detalhadas as métricas para cada componente que passou pelo *refactoring* e uma respectiva análise.

A redução de tags é a métrica que mais expressa redução de código, pois com ela é possível observar o quanto de tags evitou-se digitar. A redução de linhas serve para observar o impacto da solução no componente refatorado como um todo, porém é um índice com confiabilidade menor, pois a redução de linhas pode ser influenciada

pela forma como um desenvolvedor organiza seus códigos (quebras de linhas, tags por linha e outros). A redução de linhas HTML serve para observar o impacto da solução aplicada as linhas HTML, ou seja, a aplicação da solução na parte do código que ela foi desenvolvida para atuar.

Durante a fase de *refactoring*, alguns componentes foram melhorados, tendo sua verbosidade diminuída, possuindo também atributos adicionados, renomeados e excluídos.

Pontua-se que a etapa de desenvolvimento pode ser considerada interativa, pois durante todo o ciclo de desenvolvimento, erros foram corrigidos, testes foram reproduzidos e otimizações foram testadas e aplicadas nos componentes.

#### *1.1.3.4 Etapa de publicação*

A etapa de publicação trata do processo de publicação da solução no repositório NPM e da elaboração da documentação usada na solução. Observa-se que a etapa de publicação não foi estática, mas sim interativa com a etapa de desenvolvimento, conforme erros e melhorias eram encontradas, novas publicações foram geradas.

Para a publicação da presente solução estudou-se a forma como outras soluções efetuavam a sua publicação no repositório, além da pesquisa no Google buscando artigos que explicassem a publicação no repositório NPM.

Primeiramente ao observar a solução correlata Brightleaf Elements, descobriu-se que era possível publicar a biblioteca através da construção do pacote com alguma ferramenta *module bundler*, nas primeiras publicações da solução utilizou-se da ferramenta Rollup. Com o avanço das publicações e da etapa de desenvolvimento, optou-se por abandonar o Rollup e utilizar do transpilador Babel para efetuar o build da solução, criando um arquivo .js e um arquivo .flow para cada componente da solução.

A documentação da solução foi criada no formato GitHub Wiki, onde para cada componente é informada uma série de códigos que representam seus diversos usos e uma imagem para cada possibilidade de uso, mostrando o resultado gráfico equivalente ao código exemplificado.

# 2 FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA

No presente capítulo, são apresentados os conceitos teóricos necessários para o entendimento do presente trabalho.

## 2.1 REUSO DE SOFTWARE

A necessidade do mercado de software por produtos robustos, eficientes, tecnologicamente evoluídos e com alto poder de funcionamento está crescendo em uma taxa acelerada, porém não acompanha o tempo de desenvolvimento, forçando os produtores de software a pensar em novos design e maneiras de acelerar seus trabalhos para que as demandas solicitadas sejam entregues no tempo adequado (FAYAD,2016). Apesar da atual necessidade de otimizações, esse desejo não é novo, a busca por menor custo, menor tempo e maior qualidade acompanha o desenvolvimento de software desde o seu começo (ALVARO,2007).

Uma forma de atenuar o tempo de desenvolvimento, assim como o custo, é utilizando o reuso de software, que é o reaproveitamento do que já foi feito em detrimento de criar algo totalmente novo. Ao disponibilizar algo já criado para compor uma aplicação nova, a tendência é que o custo do software caia, o tempo de produção diminua e a qualidade aumente, tudo isso pelo fato dos componentes utilizados serem previamente testados, documentados, mantidos e codificados (FAYAD,2016).

Sametinger (1997) define dois principais grupos de benefícios do reuso de software, que são as melhorias de qualidade e a redução de esforço. Para o presente trabalho julgou-se necessário definir a qualidade, produtividade, performance, confiabilidade e redução do trabalho.

A qualidade é beneficiada pelo reuso do componente, que ao ser utilizado no decorrer do tempo, tende a ter a quantidade de erros diminuídas por sofrer constantes manutenções por parte do fornecedor (contando que o fornecedor ofereça suporte para manutenções contínuas) (SAMETINGER,1997).

A produtividade é beneficiada pela não necessidade de produção de um volume considerável de código, pois ó código utilizado do componente já foi projetado, analisado, testado e codificado previamente. Observa-se que no começo da implantação a tendência é que o tempo de produção aumente, pois ou os desenvolvedores estarão aprendendo como usar um determinado componente ou

estarão produzindo seus próprios para uso “interno” na sua organização. (SAMETINGER,1997).

O componente ao ser reutilizado diversas vezes tende a ter sua performance otimizada, isso acontece pelas possíveis manutenções na execução do componente que serão efetuadas conforme ele é usado, gerando uma constante melhoria (SAMETINGER,1997).

Assim como as melhoras na qualidade e performance, a melhora da confiabilidade é resultado dos testes e manutenções nos componentes ao decorrer do tempo de uso. Usar um software composto por unidades reutilizáveis que foram testadas várias vezes acarreta em uma confiabilidade maior de que as funções desse software serão desempenhadas corretamente. (SAMETINGER,1997).

A redução de trabalho redundante é outro benefício do reuso de software. Se ao criar uma aplicação os desenvolvedores não utilizarem do reuso, gastarão esforços desenvolvendo “partes” de software que são comuns entre diversas aplicações, e que poderiam ser reutilizadas economizando tempo e reduzindo custo (SAMETINGER,1997).

A ideia de reutilizar unidades de software para criar novas aplicações surgiu com Douglas McIlroy na conferência *Nato Software Engineering Conference* em 1968, entretanto, o desenvolvimento com a aplicação do reuso só virou um padrão entre os criadores de software no ano de 2000 (SOMMERVILLE,2011). Sommerville (2011) explica que o motivo da mudança para o desenvolvimento com reuso foi a necessidade de menor custo para produzir e manter o software, entrega-lo mais rápido e manter a sua qualidade alta. Observa-se a semelhança entre as necessidades atuais do desenvolvimento de software e as necessidades de quando o reuso virou um padrão da indústria.

As unidades de software que podem ser reutilizadas possuem diferentes níveis e diferentes tipos. Sommervile (2011, pg 296-297) define três principais tipos de unidades de software reutilizáveis, cada uma podendo variar em tipo e tamanho, são elas: reuso de sistemas, reuso de componentes e o reuso de objeto e funções.

O reuso de sistemas consiste em reaproveitar sistemas inteiros “dentro” de novas aplicações, possibilitando a criação de famílias de software (linhas de produtos de software). Uma família de software é constituída por um conjunto de aplicações

que se especializaram a partir de uma aplicação genérica, conceito esse muito semelhante aos frameworks (ver seção 2.3) (SOMMERVILLE,2011).

O reuso de componentes é baseado na ideia de reaproveitar “partes” de um software, desde o nível mais baixo como um componente individual que resolve uma pequena funcionalidade até um mais complexo que a partir da junção de diversos outros elementos forma um subsistema. Um fator possibilitado por esse tipo de reuso é o reaproveitamento de um componente/subsistema em aplicações de diferentes domínios (ver seção 2.2) (SOMMERVILLE,2011).O tipo de reuso utilizado no presente trabalho é dos componentes.

O reuso de objeto e funções se detém a um componente que fornece uma única função para ser reutilizada ou uma determinada classe para auxiliar no desenvolvimento de outra aplicação (SOMMERVILE,2011).

## 2.2 COMPONENTES

Existe uma ampla gama de definições para o conceito de componente, diversas definições são úteis, fornecendo explicações do conceito através de diferentes enfoques. Para o presente trabalho, considera-se necessário definir componente através da visão tradicional do que é um componente, abstraindo definições referentes a orientação a objetos, processos e outros. O conceito de componente é flexível a diversas interpretações por parte do projetista de componente (PRESSMAN,2011).

Segundo Pressman (2011, pg 258), componente é definido como: “Componente é um bloco construtivo modular para software de computador” nessa definição observa-se a palavra construtivo, que remete ao que compõem, algo que ajuda a construir e a palavra modular remete a separação por partes, com suas funções definidas para suprir as necessidades do contexto de cada módulo. Na visão tradicional, um componente é uma unidade constituinte de um programa que fornece uma funcionalidade. Para que essa funcionalidade seja fornecida é necessário que o componente englobe o fluxo lógico, as estruturas de dados, e uma interface para chama-lo e enviar parâmetros necessários para a atividade (PRESSMAN,2011).

Outra definição parecida com a fornecida por Pressman, é a de Sametinger (1997, pg 68, tradução nossa), que define: “Componentes podem ser vistos como parte de um sistema de software que é identificável e reutilizável”. Sametinger com sua definição revela uma característica importante dos componentes, o reuso de

software. O reuso de software e os componentes se complementam, considerando que um software pode ser “quebrado” em várias partes e cada parte é um componente, logo todos os software são construídos a partir de um fator comum, possibilitando a reutilização de uma “parte de software” criada em um sistema em qualquer outro sistema.(SAMENTINGER,1997;ALVARO,2007).

## 2.3 FRAMEWORK

Segundo Sommerville (2011, pg 300), *framework* é definido como: “Como o nome sugere, um *framework* é uma estrutura genérica estendida para se criar uma aplicação ou subsistema mais específico”.

Um *framework* possibilita o reuso de código em situações onde é interessante o reaproveitamento de uma determinada estrutura de componentes/elementos como um todo e não apenas de componentes isolados (JOHSSON,1988). Os frameworks são criados para atuar em um domínio pré-definido (SILVA,2000).

Os frameworks definem uma estrutura em um alto nível de abstração e permite que através das suas interfaces eles sejam utilizados para a criação de aplicações novas, onde os componentes resolverão problemas recorrentes entre as aplicações (FAYAD,1997). O alto nível de abstração da estrutura é essencial para o *framework*, pois separando os conceitos gerais dos conceitos específicos existentes nas aplicações é que um framework vai conseguir possibilitar que outras aplicações utilizem da sua estrutura para especializar suas funcionalidades e ideias (SILVA,2000).

Referente as características necessárias para o *framework*, Silva (2000, pg 45) define duas de essencial importância, que são a alterabilidade e a extensibilidade. A alterabilidade se refere a capacidade que o *framework* possui de alterar suas estruturas internas a partir da necessidade de adaptações por parte dos seus usuários. Isto é, o *framework* necessita separar o que é uma funcionalidade genérica (que será replicado em várias aplicações) e o que é uma funcionalidade específica (que poderá ser atualizado conforme necessidade do usuário), para que então se crie uma estrutura referente as funcionalidades gerais de modo que permita alterações nas funcionalidades específicas(SILVA,2000). A extensibilidade de um *framework* se dá pela sua capacidade de agregação de novas funcionalidades e extensão de usos (SILVA,2000).

### 2.3.1 Framework CSS

Durante a pesquisa teórica do presente trabalho, observou-se uma grande dificuldade em encontrar definições formais do termo "*framework* CSS", logo, optou-se por explicar um pouco sobre a tecnologia SASS e a sua relação com o Bulma, para apresentar o termo *framework* CSS.

O SASS é um pré-processador CSS que possui como objetivo auxiliar na escrita de código CSS, adicionado características não existentes na linguagem CSS, como variáveis e herança por exemplo (SASS,2020).

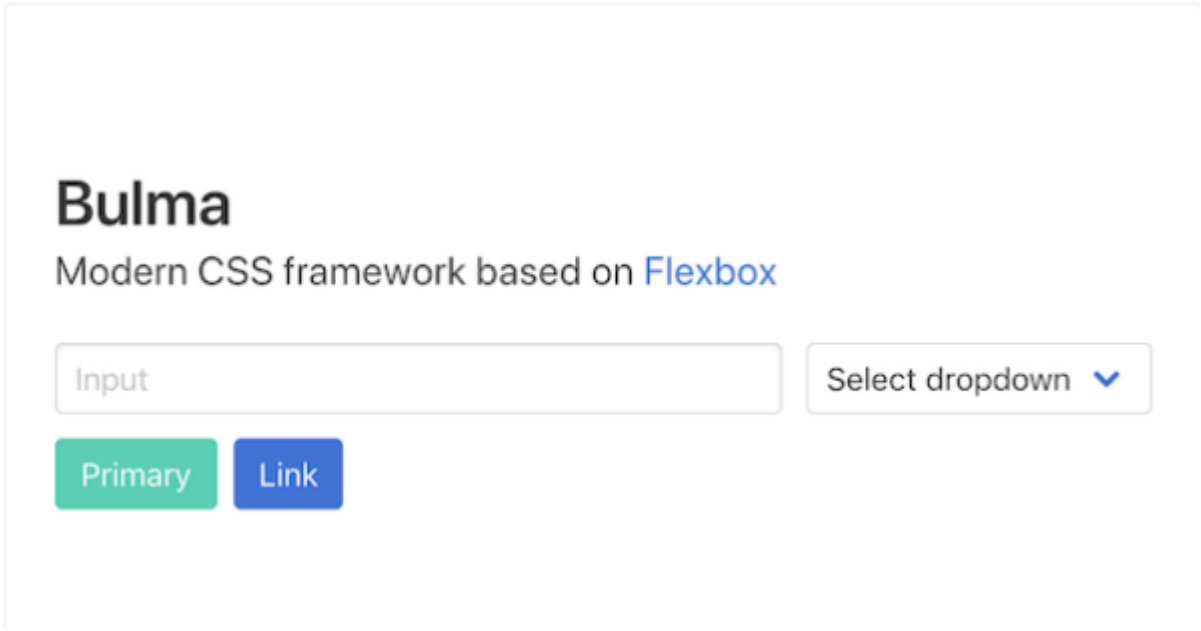
A tecnologia SASS possui um grande número de frameworks CSS construídos a partir dele, que utilizam das ferramentas do SASS para gerar seu arquivo .css ou para permitir eventuais mudanças nos seus estilos.

O Bulma é construído utilizando SASS, onde é possível alterar suas características iniciais através desse pré-processador (BULMA,2020). No presente trabalho, não é usado o Bulma como um framework completo (alterando as configurações com SASS), mas sim usado o arquivo .css que é o resultado final da construção do Bulma a partir do SASS.

Para exemplificar essa característica de customização via SASS são apresentados dois exemplos de interfaces gráficas, uma utilizando o resultado final do Bulma (arquivo .css pronto) que é o *default* da tecnologia e outro exemplo utilizando executando uma modificação com o SASS. Observa-se que o código HTML gerado pelos resultados gráficos são os mesmos na **Figura *1*** e na **Figura *2***

Conforme apresentado na **Figura *1***, existe uma tela com vários elementos que utilizam Bulma, nesse caso, o Bulma "*default*" encontrado no arquivo .css resultante do *framework*.

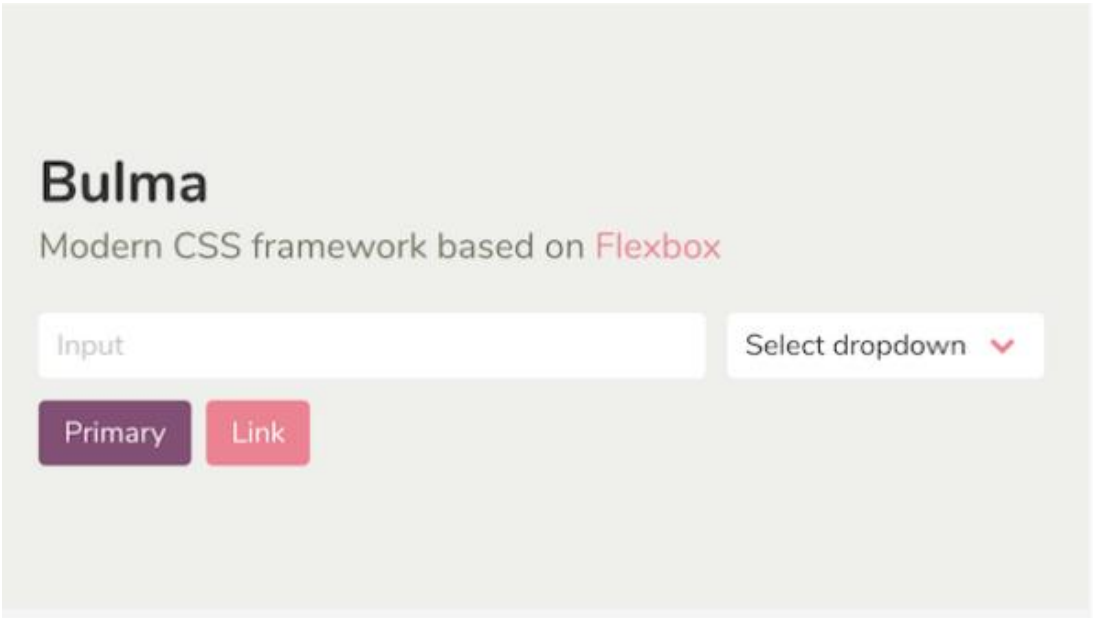
**Figura 1** - Utilização Bulma default




**Fonte**: https://bulma.io/documentation/customize/with-node-sass/

Na **Figura *2*** é possível observar uma clara mudança nas cores da interface, isso acontece pois ao invés da utilização convencional do arquivo de estilo .css fornecido para o Bulma, utilizou-se na interface customizada um arquivo .css gerado pelo SASS a partir de um outro arquivo .scss (**Figura *3***)

**Figura 2** - Utilização Bulma customizado




**Fonte**: https://bulma.io/documentation/customize/with-node-sass/

**Figura 3** - Arquivo SCSS para customização via SASS

```scss
@charset "utf-8";

// Import a Google Font
@import url('https://fonts.googleapis.com/css?family=Nunito:400,700');

// Set your brand colors
$purple: #8A4D76;
$pink: #FA7C91;
$brown: #757763;
$beige-light: #D0D1CD;
$beige-lighter: #EFF0EB;

// Update Bulma's global variables
$family-sans-serif: "Nunito", sans-serif;
$grey-dark: $brown;
$grey-light: $beige-light;
$primary: $purple;
$link: $pink;
$widescreen-enabled: false;
$fullhd-enabled: false;

// Update some of Bulma's component variables
$body-background-color: $beige-lighter;
$control-border-width: 2px;
$input-border-color: transparent;
$input-shadow: none;

// Import only what you need from Bulma
@import "../node_modules/bulma/sass/utilities/_all.sass";
@import "../node_modules/bulma/sass/base/_all.sass";
@import "../node_modules/bulma/sass/elements/button.sass";
@import "../node_modules/bulma/sass/elements/container.sass";
@import "../node_modules/bulma/sass/elements/title.sass";
@import "../node_modules/bulma/sass/form/_all.sass";
@import "../node_modules/bulma/sass/components/navbar.sass";
@import "../node_modules/bulma/sass/layout/hero.sass";
@import "../node_modules/bulma/sass/layout/section.sass";
```

**Fonte:** https://bulma.io/documentation/customize/with-node-sass/

O exemplo de customização através do SASS revela a característica de *framework* do Bulma, alterar partes de uma estrutura genérica para adaptá-la conforme necessário, sendo essas partes os seletores CSS que serão disponibilizados para utilização do usuário.

## 2.4 PROGRAMAÇÃO FUNCIONAL

A solução proposta é constituída de componentes funcionais. Um componente funcional no React é uma função que recebe parâmetros (*props*), executa procedimentos e retorna um valor, no caso do React esse valor é um trecho de código HTML.

No paradigma funcional, as funções são constituídas de procedimentos e podem depender de outras funções auxiliares para ajudar a alcançar os seus valores finais (WADLER, 1988). É possível associar a ideia de que uma função é composta de diversas outras funções com a utilização dos componentes funcionais do React. O resultado gráfico gerado por componentes funcionais do React, pode ser pensado como uma série de funções não tão complexas que são unidas para gerar um resultado mais complexo, em outras palavras uma série de componentes gráficos simples que se unem para criar um componente gráfico complexo.

Um tipo em programação funcional é um conjunto de valores que permite classificar os diversos valores envolvidos em uma função (WADLER,1988). Uma função bem formada é aquela onde os procedimentos respeitam as operações suportadas pelos tipos dos valores manipulados, por exemplo, multiplicar funções ou multiplicar números e caracteres não constituem uma função bem formada e detém grande potencial de gerar erros. A presente solução foi construída utilizando Javascript que possui tipagem fraca, ou seja, não utiliza de um controle intensivo dos tipos de cada variável, sendo essa característica em muitos casos potencializadora de erros em tempo de execução.

## 2.5 JAVASCRIPT

JavaScript é uma linguagem de programação criada no ano de 1995, sendo sua primeira versão utilizada pelo navegador Navigator 2. O JavaScript foi concebido para rodar no lado cliente, dependendo de um navegador que possui um interpretador da linguagem (SAMY,2010).

O principal objetivo para a criação do JavaScript foi o tratamento de entradas no lado cliente, tarefa antes designada ao lado servidor, o que era custoso nas tecnologias da época. A demora era grande pois para realizar o tratamento das entradas no lado cliente era necessário uma requisição ao servidor, demandando

tempo e as vezes gastando-o desnecessariamente, pois por muitas vezes algumas entradas ficavam em branco, situação que poderia ser sanada com uma linguagem que impedisse essas situações já no lado cliente (ZAKAS,2005).Apesar do objetivo primário ter sido o tratamento de entradas, a linguagem evolui e começou a interagir com praticamente todos os elementos das páginas web e com as informações exibidas por ela (ZAKAS,2005).

A característica mais importante do JavaScript é fornecer suporte para a parte comportamental das páginas web, permitindo uma maior interatividade, observando-se que o HTML por si só não consegue prover uma página web dinâmica, pois não consegue fazer o processamento de dados, necessitando de uma outra linguagem para implementar a lógica necessária (SAMY,2010). O JavaScript adicionou poder aos navegadores que antes eram simplesmente programas de exibição de documentos HTML, sem interatividade considerável (KEITH,2010).

O JavaScript consegue modificar a estrutura do documento HTML, com poder de adicionar e alterar elementos do documento, tudo isso pela capacidade que a linguagem possui de manipular a árvore DOM. O JavaScript também consegue alterar dinamicamente a estética dos elementos em conjunto com o CSS (SAMY,2010). A linguagem também oferece manipulação e processamento de dados, pois consegue interagir com formulários HTML para consultar o que foi digitado neles, calcular, tratar e realizar diversas operações sobre os dados fornecidos por usuários (SAMY,2010).

Dentre as diversas características da linguagem, é importante para o presente trabalho ressaltar que o JavaScript pode ser classificado como uma linguagem de programação funcional (ver seção 2.4). Essa classificação é possível pois a linguagem possui o poder de aplicar nas funções as mesmas operações que aplica as outras variáveis, em outras palavras, as funções possuem as mesmas características das demais variáveis (Banks e Porcello,2017).

A linguagem com o passar dos anos sofreu diversas mudanças, sendo que atualmente, com o surgimento do interpretador Node JS, o JavaScript tornou-se uma linguagem *fullstack*, onde utiliza-se ela para programar o “mais pesado” no lado servidor e a criação das páginas web no lado cliente, fornecendo soluções mais abrangentes do que os objetivos iniciais no cerne da sua criação (Banks e Porcello,2017).

### 2.5.1 FLOW

A linguagem JavaScript possui tipagem fraca, onde uma variável pode assumir diversos valores com tipos diferentes e pode a mesma variável aceitar diferentes tipos no decorrer da execução do script (com tanto que essas variáveis não sejam constantes) (SAMY,2010).

Conforme apresentado na seção 2.4, os componentes da presente solução são funcionais e a definição correta de tipos é importante para assegurar a qualidade da função, evitar que erros básicos aconteçam.

Ao analisar a questão de qualidade das funções e da tipagem fraca do JavaScript, optou-se por adicionar um checador de tipos chamado Flow que visa incrementar a qualidade da solução proposta, pois ao controlar os tipos as chances de erros lógicos se tornam menores.

O Flow é um checador de tipos que permite a identificação de tipos errados em tempo de compilação, possuindo como um dos objetivos permitir um código JavaScript mais confiável (FLOW, 2020). O Flow funciona através de anotações no código que permite o desenvolvedor determinar qual o tipo de cada elemento do seu código (FLOW,2020).Conforme mostrado na **Figura *4***, o desenvolvedor informou que o parâmetro n é do tipo *number*, logo, se a função *square* for invocado e receber como parâmetro algum valores que não seja número, o Flow avisará em tempo de compilação (conforme mostrado na linha 6), ajudando o desenvolvedor a identificar potenciais erros no código e com isso melhorar a confiabilidade do seu código.

**Figura 4** - Exemplo Flow

```
1  // @flow
2  function square(n: number): number {
3    return n * n;
4  }
5
6  square("2"); // Error!
```

**Fonte**: Flow,2020

## 2.6 HTML

O HTML (*HyperText Markup Language*) foi criada no CERN (Organização Europeia para a Pesquisa Nuclear), que desenvolveu a linguagem com o objetivo de facilitar a troca de documentos eletrônicos na internet, possibilitando o envio e recepção de trabalhos completos que possuíam o poder de empacotar diversos formatos de informações como texto, sons e imagens. Outro facilitador criado com o HTML foi o da interligação de documentos, a capacidade de a partir um documento na Internet, navegar para diversos outros documentos, constituindo o chamado hipertexto (MUSCIANO & KENNEDY,2000)

Segundo Musciano e Kennedy (2000, pg 9, tradução nossa), o HTML é definido como: “HTML é uma linguagem de layout de documento e especificação de hiperlink”. A linguagem estrutura a página web, determinando como ela será exibida, como estão organizadas as informações e possibilita uma forma de interatividade entre documentos (não confundir com interatividade da página possibilitada pelo JavaScript), através dos links de hipertexto.

### 2.6.1 Estrutura do HTML

Um documento HTML é formado essencialmente por textos, que são as informações contidas em uma página web e por tags que determinam a estruturação do documento (W3,2020).A estrutura do HTML, da maneira mais simples, é composta essencialmente por uma tag <html> que incorpora duas outras tags, que são <head> e <body> (MUSCIANO & KENNEDY,2000). Com a tag <head> é possível definir metadados para a página web, esses metadados fornecem informações sobre os dados da página, exemplos dessas informações são a estrutura de caracteres da página, o estilo CSS usado e o título, todos eles descrevendo informações sobre o documento HTML, sobre a página(W3,2020). Na tag <body> é colocado o texto que será exibido na página web, assim como as demais tags que determinam como o texto será exibido ao usuário, além de permitir exibição multimídia (imagens, sons entre outras mídias) e vinculação de hiperlinks (MUSCIANO & KENNEDY,2000).

Abaixo um exemplo de código simples no HTML, onde é possível visualizar a estrutura básica de um documento HTML. As tags obedecem a estrutura antes definida, onde <html> engloba <head> e <body>, tendo <head> uma outra tag que define metadados e a <body> o texto que será exibido para o usuário.

**Figura 5** - Código Simples HTML

```
<html>
<head>
<title>Barebones HTML Document</title>
</head>
<body>
This illustrates, in a very <i>simp</i>le way,
the basic structure of an HTML document.
</body>
</html>
```

**Fonte:** Musciano e Kennedy,2000.

### 2.6.2 Sintaxe básica do HTML

A sintaxe do HTML é composta por tags, essas tags são formadas através do nome da tag que se deseja usar (fornecida pela linguagem HTML) entre colchetes. As tags em HTML não são *case-sensitive*, ou seja, informar o nome com partes minúsculas ou maiúsculas não interfere no funcionamento, com tanto que sintaxe da palavra esteja correta. As tags do HTML suportam a inclusão de atributos, eles são passados antes do primeiro colchete e servem para alterar características da tag ou do seu conteúdo. A passagem de valores para atributos segue uma estrutura de nome do atributo que será alterado, seguido pelo símbolo “=” e o novo valor que o atributo assumirá. Caso um novo valor informado ao atributo não contenha espaços em branco, pode-se passar o valor após informar o “=”, caso contrário deve-se passar o valor entre aspas (simples ou duplas) (MUSCIANO & KENNEDY,2000).

## 2.7 CSS

No começo da Internet (1990-1993), o HTML era usado como uma linguagem que criava várias partes de um documento, possuindo uma característica essencialmente estrutural para as páginas web, sendo usado apenas para “montar” um documento que era exibido através do navegador, não sendo enfoque principal da linguagem a parte estética (MAYER,2004).

Com a percepção do potencial da Internet, existiu um aumento considerável no número de sites criados, diversas empresas e usuários comuns começaram a criar suas páginas web. Com esse aumento, os criadores de sites desejavam o poder de

adicionar determinados efeitos para suas páginas, adicionar a parte estética ao conteúdo estrutural já suportado pelo HTML. Para suprir a necessidade por uma melhor estética, foi adicionado ao HTML determinadas tags que ajudariam nessa parte, provendo características estéticas para uma linguagem estrutural (MAYER,2004). No entanto, adicionar essas características ao HTML, fere o princípio básico da linguagem, de não fornecer aos dispositivos que o interpretam informações sobre a parte de apresentação da página (cores, fonte, posicionamento e outros atributos estéticos) (SAMY,2012).

Com a mistura da parte estrutural com a parte de apresentação, um determinado problema surgiu, a falta de semântica para as marcações estéticas do HTML. Com as extensões criadas no HTML, as páginas começaram ser desenvolvidas de modo que não era possível separar estruturas específicas de texto comum, podendo uma estrutura como um título ser representada tanto de forma estrutural com a tag <h1> como de forma textual utilizando a tag criada para estética, como a tag <font> por exemplo, fazendo do título apenas um texto comum. Os principais problemas causados pela falta de semântica introduzida são a extrema dificuldade de indexação, redução da acessibilidade, dificuldade de manutenção de código e dificuldade para criar páginas de apresentação avançadas (MAYER,2004)

Para solucionar os problemas gerados pela mistura da parte estrutural com a parte estética no HTML, o W3 oficializou como recomendação a utilização do CSS para atender a parte estética necessária para criação de páginas web, no ano de 1996 (MAYER,2004). O CSS (Cascading Style Sheets), pode ser definido como:

> Com CSS, você pode controlar a cor, a fonte, o tamanho do texto, o espaçamento entre os elementos, como os elementos são posicionados e dispostos, quais imagens ou cores de plano de fundo serão usadas, diferentes telas para diferentes dispositivos e tamanhos de tela (**W3,2020,tradução nossa**).

O CSS possui como finalidade permitir que linguagens de marcação, como o HTML, cumpram com seu objetivo primário, funcionar como uma linguagem apenas de estruturação de conteúdo e marcação (SAMY,2012).

### 2.7.1 Sintaxe Básica do CSS

Para que o CSS possa fornecer suporte ao HTML, adicionando a parte estética, o mínimo que precisa ser usado é uma Regra CSS, que é a menor quantidade de código possível para se estilizar algum elemento HTML (SAMY,2012).

**Figura 6** - Sintaxe da Regra CSS


**Fonte:** Samy,2012.

Os componentes de uma Regra CSS são:

1. **Seletor** - Define qual elemento será estilizado com a declaração, pode-se escolher todos os elementos de uma determinada estrutura. Um exemplo de utilização de seletor é utilizando a estilização para o componente <p> que representa pedaços de textos dentro da página web, ao se adicionar um seletor com um estilo para <p> todo o conteúdo entre tag <p> será estilizado por uma regra CSS (SAMY,2012)..
2. **Propriedade** - Atributo que será estilizado, qual característica do elemento HTML será alterada (SAMY,2012).
3. **Valor**: Valor que a propriedade assumirá para que seja possível estilizar o elemento (SAMY,2012).
4. **Declaração**: Abrange os componentes propriedade e valor (SAMY,2012).

Um exemplo de um simples código CSS é mostrado na **Figura 7**, onde é descrito um seletor que é “button”, uma propriedade “background-color” que é responsável por armazenar qual a cor do botão e o valor “aquamarine” que refere-se a cor propriamente dita que terá os botões que usarem esse estilo CSS. Na **Figura 8** é mostrado o código utilizado para criar um botão HTML e na **Figura 9** o resultado gráfico desse do código, que por ser um botão será exibido com a cor de fundo “Aquamarine”. Observa-se que o nome do arquivo que contém o código descrito na

**Figura 8** se chama example_css.css, logo é esse código é referenciado na **Figura 8** (“import ‘./css/example_css.css”).

**Figura 7** - Exemplo de código CSS.

```
button{
    background-color: aquamarine;
}
```

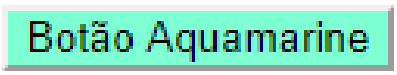
**Fonte:** O próprio Autor, 2020

**Figura 8** - Código Botão Aquamarine.

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';
import './css/example_css.css';

ReactDOM.render(
    <button>Botão Aquamarine</button>
  , document.getElementById("root"))
```

**Fonte:** Próprio Autor, 2020

**Figura 9** - Resultado Gráfico Botão Aquamarine




**Fonte:** Próprio Autor,2020

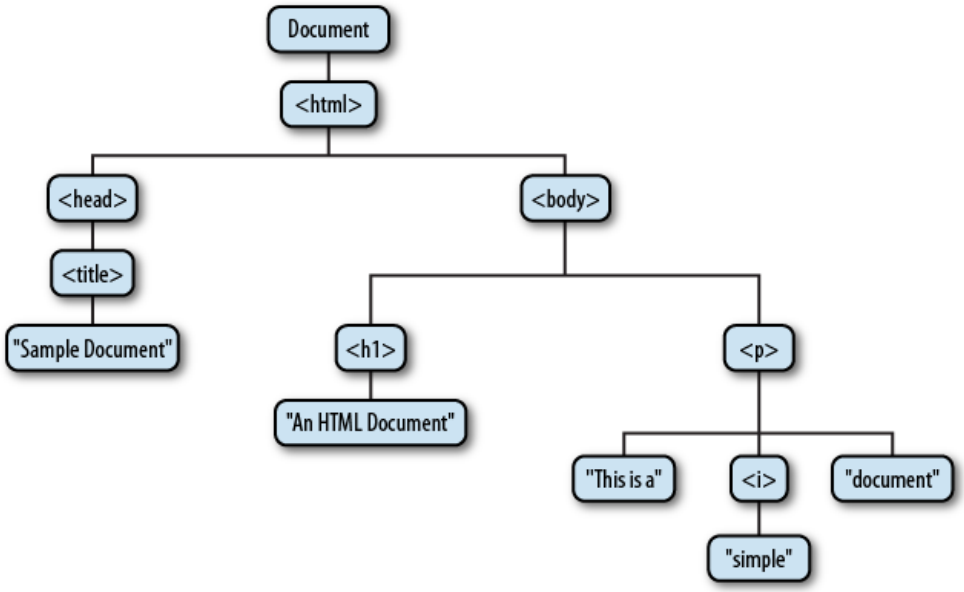
## 2.8 DOM

O DOM (*Document Object Model*), é uma interface que permite acessar, deletar, adicionar e remover elementos de documentos HTML e XML. O DOM organiza os elementos HTML em um formato hierárquico semelhante ao de uma árvore (estrutura de dados) e define como as operações podem ser realizadas nos elementos e os métodos que são fornecidos para tal (W3,1998).Cada nó da árvore pode representar uma tag HTML, um comentário ou um texto contido entre tags HTML(FLANAGAN,2011).

A nomenclatura dada (Document Object Model) foi assim escolhida inspirada na orientação a objetos, pois um documento representado por uma árvore DOM abrange não somente os dados dos objetos (nós), mas também seus comportamentos. Sendo o nó dá árvore DOM representada por um objeto, esse objeto possui uma interface para consulta e a manipulação, uma forma de acessar os dados e ações do objeto e a existência de relações entre os objetos da árvore(W3,1998).

A **Figura *10*** mostra uma estrutura lógica criada pelo DOM para um determinado código, onde cada tag constitui um nó e assim como o texto presente entre as tags, constituem outros nós.

**Figura 10** - The tree representation of an HTML document.




**Fonte**: Flanagan(2011)

A terminologia utilizada para expressar a relação entre os nós é baseada em uma árvore familiar, onde os primeiros nós abaixo de outro nó são considerado filhos daquele nó, o primeiro acima é considerado pai e qualquer nível acima é considerado ancestral e qualquer nível abaixo é considerado descendente (FLANAGAN.2011).

Um dos principais objetivos presente na especificação do DOM, é de fornecer uma interface padrão que permita adicionar dinâmica ao HTML em diversos ambientes, permitindo que uma linguagem como o JavaScript possa criar páginas web

dinâmicas em diversos navegadores, fornecendo interoperabilidade para as linguagens que suportam processamento na web. (W3,1998).

## 2.9 REACT

O React é uma biblioteca criada no ano de 2013 pelo Facebook, com o intuito de auxiliar no desenvolvimento de interfaces de usuários na web (Banks e Porcello, 2017). O React não é um *framework*, mas sim uma biblioteca, não sendo objetivo da tecnologia fornecer o aparato necessário para construção completa de qualquer aplicação, sendo o React uma biblioteca especializada em apenas uma função, construção de interfaces gráficas para páginas web (BANKS & PORCELLO,2017).

O React é baseado em componentes, onde componentes de diversas complexidades controlam seus estados e reagem as suas modificações de forma independente. Essa característica do React permite a composição de interfaces gráficas, onde componentes menores são compostos para criação de uma interface gráfica complexa, ou um componente React mais complexo (conceito apresentado também na seção 2.4).

Uma das características que tornam o React atrativo aos desenvolvedores é a maneira como ele manipula a árvore DOM, através do conceito VirtualDOM, sendo esse assunto tratado no APÊNDICE A – VIRTUAL DOM**.**

### 2.9.1 JSX

Segundo Banks e Porcello (2017, pg 81, tradução nossa), o JSX é: “Uma extensão do JavaScript que nos permite definir elementos React usando uma sintaxe similar ao HTML”. O JSX é uma extensão da sintaxe do JavaScript, que auxilia no desenvolvimento utilizando React, permitindo utilizar código HTML “dentro” do JavaScript. Apesar de não ser obrigatório o uso do JSX para se programar com React, alguns benefícios são alcançados com a extensão da sintaxe. A facilidade obtida ao escrever HTML dentro do JavaScript e alocação de elementos no DOM (eliminando uso de alguns métodos do DOM) são exemplos de benefícios encontrados com o uso do JSX (W3,2020; REACT,2020).

A sintaxe JSX permite a inclusão de qualquer expressão JavaScript dentro do código HTML, ou seja, é possível passar valores comuns, como uma cadeia de

caracteres pura, assim como é possível informar funções, elementos React e código HTML(W3,2020; REACT,2020).Outra facilidade oferecida pelo JSX é a possibilidade de não precisar se preocupar tanto com a sintaxe das funções, não necessitando do controle comum que se deve ter com parênteses e chaves (STEFANOV,2016).

**Figura 11** - Exemplo Table JSX

```
const Table = (props: Props) => {
    return assembleTable(props);
}

function assembleTable(props) {
   var definition = (props.definition ==  undefined)? "table": "table " + props.definition;

    var code =  <table className={definition}>
                    {isArrayOk(props.itens_header) && assembleHeader(props)}
                    {isArrayOk(props.itens_body) && assembleBody(props.itens_body,props.onClickRow)}
                    {isArrayOk(props.itens_footer) && assembleFooter(props.itens_footer)}
                </table>

    return code;
}
```

**Fonte**: O próprio autor, 2020.

Na **Figura 11** é apresentado um exemplo de código onde as expressões JSX são utilizadas para “montar” o HTML de um componente, ou nesse caso, montar o próprio componente. A tag <table> especificada na imagem será a base do componente, “dentro” dela é colocado todo o código necessário para gerar um componente completo, toda esse código será obtido através das três funções especificadas, e o resultado das funções será posto em ordem dentro da tag <table> através de expressões JSX especificadas pelas chaves que cercam as funções.

Como é possível ver na **Figura 11**, o JSX assemelha-se muito com o HTML e de fato, o JSX é uma extensão extremamente parecida com o HTML, porém com facilidade de informar código JavaScript entre chaves. Algumas das principais diferenças sintáticas do JSX para o HTML são que no JSX todas as tags devem ser fechadas, o atributo “class” do HTML é equivalente ao atributo className no JSX, ao informar um estilo, o atributo “style” da tag recebe um objeto e não uma cadeia de caracteres e o nome das propriedades CSS informadas são no formato camel case e não separados por vírgulas igual o HTML tradicional.

### 2.9.2 Componente React

Os componentes *React* são as partes que compõem uma interface de usuário, sendo um componente *React* uma parte integradora de um todo (no caso, uma interface gráfica) e essa parte pode ser reaproveitada em diversas outras interfaces gráficas, fornecendo reuso de software (BANKS e PORCELLO,2017).

Os componentes React permitem uma grande divisão e reaproveitamento de partes de interfaces comuns como por exemplo botões, tabelas, painéis, campos de texto e etc. Ao permitir tamanha divisão, o React possibilita o isolamento de todas essas partes descritas, fornecendo a possiblidade de reaproveita-las ao longo de todo o desenvolvimento das aplicações.

Um componente React possui como saída/resultado um trecho de código HTML que será renderizado pelo navegador (REACT,2020). O React fornece a possiblidade de composição, isto é agrupar diversos componentes React de diferentes complexidades e funções em um único componente (assunto tratado também na seção 2.4 e 2.9), essa composição pode ser pensada como a união de vários pequenos trechos de código HTML, que podem formar uma grande quantidade de código final. A composição é possibilitada pelo atributo *Children* oferecido pelo React (conceito apresentado em 2.9.3) (REACT,2020).

### 2.9.3 Children no React

O React possui uma característica muito utilizada no desenvolvimento da presente solução, que é o atributo *children* encontrado nas *props*.

Conforme apresentado em 2.9.2, o React fornece a possibilidade de composição. Para que a composição seja possível, é necessário a utilização do atributo children de props. O atributo children permite a criação de uma hierarquia de componentes React, onde um componente React pai utiliza de um ou mais componentes React filhos para existir.

Um exemplo da utilidade da hierarquia é ao se pensar em uma interface gráfica onde existem os componentes React Columns, Column, Container, Button e Field. É possível que ao construir uma interface com os componentes antes citados, utilize-se Columns como pai, Column como filho de Columns, Container como filho de

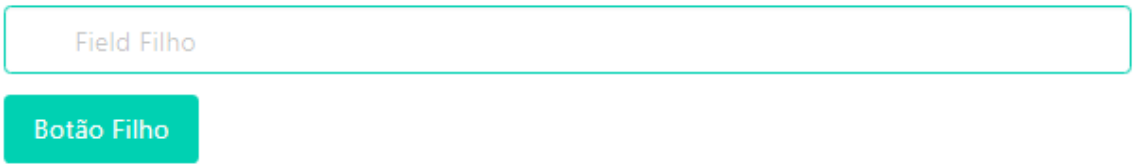
Column e Button e Field como filhos de Container. Com essa hierarquia é possível a criação de uma interface gráfica onde a partir de Columns existiu uma composição de diversas pequenas partes gráficas para a criação de uma única e mais complexa interface gráfica, conforme apresentado na figura **Figura 13**. No código apresentado na **Figura 12** é possível observar que a partir de Columns vários outros componentes foram adicionados como filhos, e isso só foi possível devido a existência de *children*.

**Figura 12** - Código exemplo do uso de *children*

```
<Columns definition="is-centered">
    <Column definition="is-half">
        <Container>
            <Field placeholder="Field Filho" input_definition="is-primary"/>
            <Button definition="is-primary"> Botão Filho</Button>
        </Container>
    </Column>
</Columns>
```

**Fonte**: O próprio autor, 2020

**Figura 13** - Resultado Gráfico do exemplo de children




**Fonte**: O próprio autor, 2020

### 2.9.4 Classificação

Da classificação dos componentes quanto ao seu estado, existem duas possibilidades, a do componente possuir um estado(*statefull*) ou não(*stateless*). Um componente com estado é aquele que possui dados que determinam a versão dele em determinado tempo, já um componente sem estado é aquele que não possui esses dados.

Antes da versão 16.8 do React as manipulações envolvendo o estado de um componente eram possibilitadas apenas através das classes. Na versão 16.8, a biblioteca recebeu a adição dos chamados *hooks*, que são funções especiais criadas para facilitar as operações de criação e alteração do estado de um componente

(REACT,2020). Observando-se a grande mudança existente, optou-se por apresentar a classificação dos componentes ao utilizar a abordagem utilizando classes (mais próxima da orientação a objetos) e a abordagem funcional, que é a utilizada na presente solução.

#### *2.9.4.1 Utilizando classe*

Um componente *stateless* definido através de classe não apresenta um objeto *state* explicitado na sua codificação, conforme mostrado na **Figura 14**, sendo a única possibilidade de dinâmica oferecida pelo atributo *props,* fazendo o componente dependente de informações do seu invocador ou componente pai caso queira adicionar dinâmica a sua renderização.

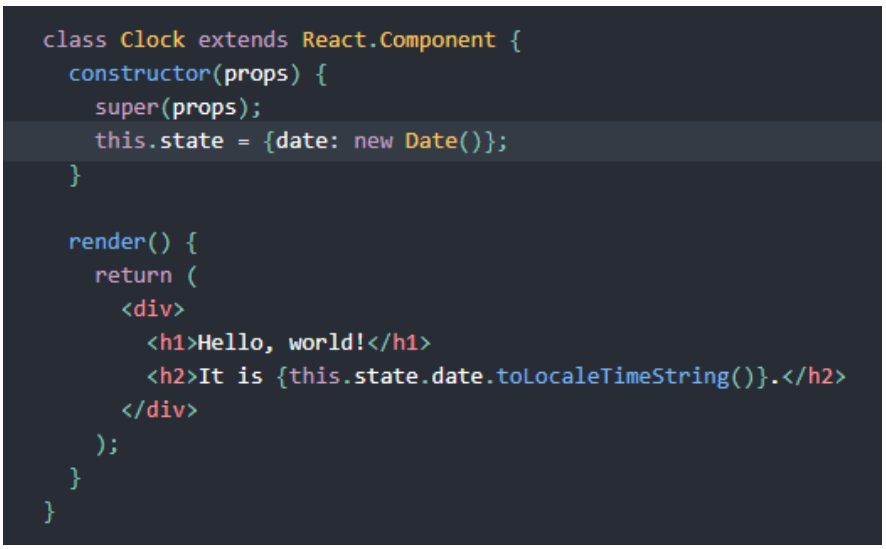
**Figura 14** - Componente *stateless* utilizando classe

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

**Fonte**: https://pt-br.reactjs.org/docs/state-and-lifecycle.html

Um componente *statefull* definido através de classe apresenta um objeto state explicitado no seu código, sendo o próprio componente que predefine os atributos que constituem o seu estado (que encapsula o estado), conforme mostrado na ***Figura 15***

**Figura 15** - Componente statefull utilizando classe



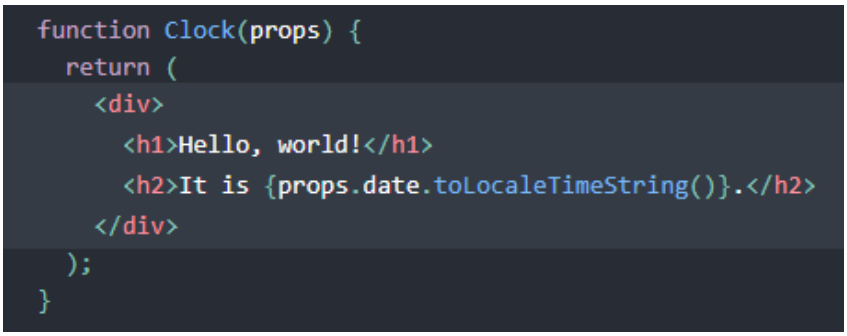
```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

**Fonte**: https://pt-br.reactjs.org/docs/state-and-lifecycle.html

### *2.9.4.2 Utilizando funções*

Os componentes React podem ser componentes funcionais, para isso basta que o componente seja uma função que retorne um trecho de código HTML, conforme mostrado na **Figura 16**. Para a criação de um componente funcional *stateless* não é necessário o uso dos *hooks*, entretanto, caso exista e necessidade de manipular o estado de um componente funcional é necessário a adição de algum *hook*. No exemplo da **Figura 17** é possível observar o uso do *hook useState* que recebe um valor inicial 0 e retorna duas variáveis, que são *count* e *setCount.* A variável *count* representa um atributo que compõem o estado do componente *Example* e a variável setCount é uma função que poderá alterar o atributo *count.* Observa-se que os exemplos são diferentes na **Figura 16** e **Figura 17** propositalmente.

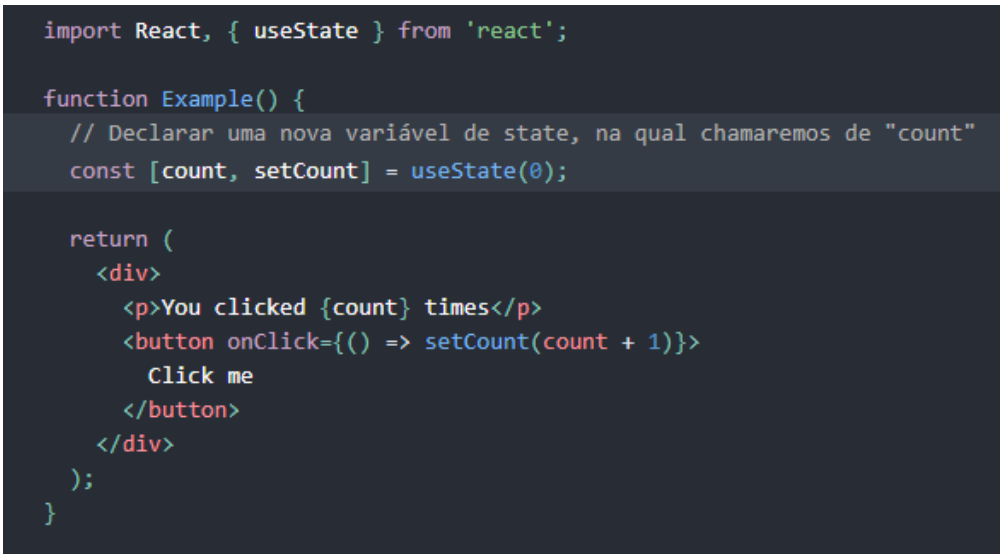
**Figura 16** - Componente *stateless* utilizando funções



```
function Clock(props) {
  return (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {props.date.toLocaleTimeString()}.</h2>
    </div>
  );
}
```

**Fonte**: https://pt-br.reactjs.org/docs/state-and-lifecycle.html

**Figura 17** - Componente statefull utilizando funções



```
import React, { useState } from 'react';

function Example() {
  // Declarar uma nova variável de state, na qual chamaremos de "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

**Fonte**: https://pt-br.reactjs.org/docs/hooks-state.html

### 2.9.5 Props e State

O *state* de um componente React, representa o estado em que o mesmo se encontra e pode ser composto por diversos atributos. Os atributos que representam um estado são previamente definidos no React e ao receberem novos valores resultam em uma atualização, uma renderização nova do componente que tende a trazer consigo novos resultados visíveis na página web, como uma troca de cor, troca de texto e etc. O estado de cada componente é privado e sua informação flui de cima para baixo na hierarquia de componentes React. O componente pai pode informar seu estado via propriedade para seus componentes filhos, entretanto os filhos não saberão de qual componente veio esse estado, essa característica permite maior

reuso, pois permite que um componente possa virar filho de diversos outros componentes (REACT,2020). Observa-se que essa questão de gerência de estado é tratada também na seção 4.1.3.

O atributo *props* presente nos componentes React representam dados que são passados pelo invocador do componente, dados enviando no momento em que um componente React é chamado para renderização (e posteriormente atualizações). A principal função das props é trazer para um componente *React* dados necessários para seu funcionamento, para alguma função, atribuição e outras necessidades de dados que o componente possua.

As *props* são imutáveis, isso significa que ao informar um valor de props para um componente, internamente os valores de props não irão mudar, entretanto é possível utilizar do gerenciamento de estados para dar dinâmica a um componente (REACT,2020).

#### *2.9.5.1 Exemplo de uso Props e State*

Na **Figura 18** e **Figura 19**, é mostrado um exemplo do uso estático de *props*. O componente InputParent possui a função de retornar o filho Input com um código onde a *props* é passado do pai ao filho e irá definir o estilo do *element* <input>. No exemplo o estilo de <input> jamais mudará, pois o seu componente pai, está enviando estaticamente o nome da característica estética do seu filho. Pode-se com esse exemplo, entender que as *props* são imutáveis, elas não devem mudar dentro da implementação do Input, e não é possível mudar o valor das props que o Input receberá se não for por meio do uso do *state*.

**Figura 18** - Exemplo de componente sem estado

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.css';

const Input = (props) => {
  return <div className="columns is-centered">
    <div className="column is-half">
      <input className={props.definition} value="Example Props and State" />
    </div>
  </div>
}

const InputParent = (props) => {
  return <Input definition="input is-primary" />
}

ReactDOM.render(<InputParent />, document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

**Figura 19** - Resultado gráfico de InputParent.


**Fonte**: O próprio autor, 2020

Na **Figura 20** e **Figura 21** é apresentado um exemplo um pouco mais complexo de como “alterar” os valores das props utilizando do estado do componente pai.

A diferença do exemplo da **Figura 20** e **Figura 21** e do anterior(**Figura 19**) é basicamente que a *props* definition agora ao invés de ser informada estaticamente de pai para filho, é informada utilizando o estado do pai que é dinâmico e que pode ser alterado via função *onClick*. Com essa alteração, cada vez que o evento *onClick* alterar o estado do componente pai, essa alteração provocará uma nova renderização do componente pai que provocará uma nova renderização do componente filho, passando o seu novo valor do estado como props, revelando a característica antes mencionada, de manipular o estado do filho a partir do estado do seu pai. Observa-se que nesse exemplo também foi adicionado um elemento button que é apenas uma forma de disparar o evento *onClick* e também adicionado elementos do Bulma, esses para fins estéticos do exemplo. O funcionamento de *onClick* é exemplificado na **Figura 21** onde é possível observar que a cada clique que o componente Button sofrer, uma

nova cor será atribuída a ele e um novo nome, e cada alteração uma nova renderização é executada.

**Figura 20** - Exemplo de componente com estado

```
const Input = (props) => {
  return <div className="columns is-centered">
    <div className="column is-half">
      <input className={props.definition} value="Example Props and State" />
    </div>
    <div className="column is-half">
      <button className="button" onClick={props.onClick}>{props.name}</button>
    </div>
  </div>
}

const InputParent = (props) => {
  const [state, { onClick }] = useModel();
  return <Input definition={state.definition} onClick={onClick} name={state.name} />
}
```

**Fonte**: O próprio autor,2020

**Figura 21** - Função useModel de InputParent

```
23  function useModel() {
24    var initial = {
25      definition: "input is-success",
26      count: 0,
27      name: "Trocar para primary"
28    }
29
30    const [state, setState] = useState(initial);
31
32    var onClick = () => {
33      if (state.count == 0) {
34      setState({definition: "input is-primary", count: state.count + 1, name: "Trocar para info"})
35      } else if (state.count == 1) {
36        setState({definition: "input is-info", count: state.count + 1, name: "Trocar para warning"})
37      } else if (state.count == 2) {
38        setState({definition: "input is-warning", count: state.count + 1, name: "Trocar para danger"})
39      } else if (state.count == 3) {
40        setState({definition: "input is-danger", count: state.count + 1, name: "Fim das trocas"})
41      }
42    }
43
44    return [state, { onClick }]
45  }
46
47  ReactDOM.render(<InputParent />, document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

## 2.10 BULMA

O Bulma é um framework CSS, de código aberto que auxilia os desenvolvedores na manipulação da parte estética das interfaces gráficas de aplicações web, sendo as principais características dele ser responsivo, modular, baseado no Flexbox e sem custo para uso (BULMA,2020).

- Responsivo: Significa que a estética de elementos que utilizam Bulma se adaptam a diferentes dispositivos, tamanhos de telas e janelas. Essa característica permite que o desenvolvedor ao utilizar o Bulma não precisa se preocupar tanto com alterações de layout que ocorrem de um ambiente para outro ou gastar tempo elaborando métodos para adaptar um elemento a diferentes tamanhos de tela (BULMA,2020).

- Modular: O Bulma fornece a opção de escolher quais das suas classes usar sem precisar importar todo o arquivo .css para o projeto, se desejado é possível importar somente os seletores referentes a um determinado componente (BULMA,2020).

- Flexbox: O Flexbox é um conjunto de propriedades fornecidas pela implementação do CSS que facilita a construção de interfaces responsivas(W3,2020). O Bulma na sua implementação utiliza dessas propriedades para construir seus seletores, fornecendo aos elementos poder de adaptação do layout em diferentes ambientes (BULMA,2020).

- Grátis: O Bulma é totalmente livre para uso, possuindo seu código acessível no GitHub, bastando apenas baixa-lo e utilizá-lo (BULMA,2020).

A estrutura do Bulma é um arquivo CSS (.css) com vários seletores que permitem as modificações estéticas nos elementos da página web, bastando para o desenvolvedor apenas importar o Bulma e utilizar os seus seletores para designar a classe de tags HTML.

Na **Figura 22** é possível observar que para criar um botão com características do Bulma, basta informar na tag <button> o atributo className com o valor “button”, que faz referência ao arquivo .css e irá fornecer a estética requerida. Pontua-se que “is-warning” encontrado no atributo *className* serve para dar a cor amarelada ao botão.

**Figura 22** - Exemplo Botão utilizando Bulma

```
<button className="button is-warning">Button Warning</button>
```

**Fonte**: O próprio autor,2020

**Figura 23** - Exemplo botão Bulma


**Fonte**: O próprio autor,2020

O Bulma possui 5 grupos principais de classes que são Columns, Layout, Form, Elements e Components

O grupo Columns possui dois elementos que são Columns e Column. Esses dois elementos possuem como objetivo adicionar responsividade a interface gráfica, pois Columns e Column permitem que os elementos “respondam” a diferentes layout e tamanhos de interface, característica essa possibilitada pelo FlexBox.

O grupo Layout fornece elementos para tratar a estrutura da página web, fornecendo uma série de possibilidades de como organizar cada elemento, fornecendo elementos que possibilitam diversas estruturas para os elementos.

O grupo Form fornece os elementos básicos de entrada de dados.

O grupo Elements descreve os elementos essenciais, trata-se de pequenas partes de interface independentes, que podem ser usadas para compor interfaces mais complexas.

O grupo Components fornecem os elementos Bulma que são resultados da composição de outros elementos Bulma, possuindo complexidade mais elevada.

# 3 SOLUÇÕES EXISTENTES

Buscando entender o funcionamento de soluções usadas para apoiar o desenvolvimento utilizando React e o framework CSS Bulma, pesquisou-se diversas soluções já existentes, que estão alinhadas aos objetivos do presente trabalho. O presente tópico apresenta as soluções encontradas, fornecendo um panorama geral de uso e pontos importantes analisados. Primeiramente é explicado quais foram as características que nortearam a busca e em seguida é detalhada cada solução correlata.

## 3.1 CARACTERÍSTICAS DAS SOLUÇÕES

Para nortear a busca pelas soluções existentes, definiu-se cinco características de uma solução correlata, são elas:

- A solução deve utilizar o React para criação dos componentes.
- A estética fornecida pelos componentes deve ser construída a partir do framework CSS Bulma.
- A solução contém implementação, possui a codificação dos componentes disponível para estudo
- A solução possui uma quantidade de componentes adequada, fornecendo no mínimo 10 componentes para utilização.

### 3.1.1 Pesquisa das soluções

As pesquisas em ferramentas como *Google Scholar* não revelaram muitos benefícios para o presente trabalho, os resultados refletiam guias didáticos, tutoriais e artigos que detalhavam as tecnologias, porém não forneciam uma série de componentes construídos com os objetivos desejados.

Para suprir a “carência” fornecida pelas ferramentas acadêmicas, optou-se por procurar em repositório mais “informais”, como o NPM e a busca tradicional no *Google*.

A busca tradicional através do *Google* não retornou resultados interessantes, sempre retornando o mesmo pacote e indicando tópicos que não supriam os pré-requisitos definidos;

O NPM é um repositório JavaScript *open source* que possui mais de 1 milhão de pacotes JavaScript disponíveis para uso. Ao pesquisar no NPM, o intuito foi descobrir soluções existentes criadas utilizando React e Bulma, pois a busca no *Google* não retornava mais que um ou dois resultados interessantes.

No repositório NPM, foram submetidas algumas pesquisas, listadas na tabela abaixo.

**Tabela 1** - Pesquisa no NPM

| Sequência de palavras pesquisadas | Pacotes Encontrados |
|---|---|
| "react";"bulma";"components" | 39 |
| bulma;"components" | 139 |
| react;"bulma" | 89 |

**Fonte**: Próprio Autor, 2020

O site do NPM onde as soluções foram pesquisadas, fornece 3 percentuais sobre cada pacote JavaScript, que são: popularidade, qualidade e manutenção. Optou-se por observar as soluções com qualidade maior que 90% e desconsiderou-se a popularidade e manutenção. De todos os pacotes retornados, muitas soluções se mostraram inviáveis para utilização no presente trabalho, por possuírem acoplamento com outras tecnologias que não estão no presente escopo (como Vue.js) ou por solucionar apenas um pequeno conjunto de componentes e não uma solução mais complexa e completa.

Analisando o código de uma série de soluções e considerando os requisitos pré-definidos, selecionou-se 2 trabalhos a partir do NPM. Os trabalhos escolhidos se chamam *Brightleaf Elements* e *React Bulma Components*.

#### *3.1.1.1 Brightleaf Elements*

*Brightleaf Elements* é um conjunto de elementos *React* inspirados no *Bulma* que possui como objetivo auxiliar na criação de aplicativos e sites, sendo construída

por um usuário do GitHub chamado Kevin (sem mais informações encontradas).A solução engloba 4 grandes conjuntos de classes do Bulma, que são layout, forms, elements e components, excluindo apenas o conjunto columns. Observa-se que a versão da solução, ao ser analisa no presente trabalho, é a 1.9.0 (commit em 29/12/2019), assim como sua documentação fornecida em https://brightleaf.dev/elements/#/.

#### 3.1.1.1.1 Propriedades estéticas

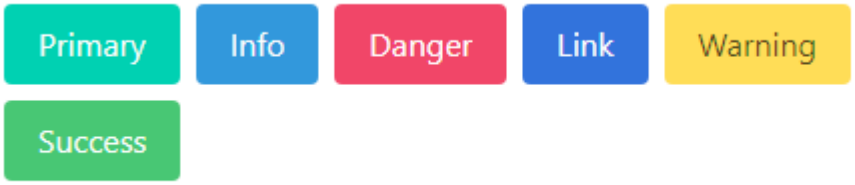
Para modificar a estética de um componente React através da presente solução correlata, basta informar uma propriedade (*props*) que represente uma classe existente no Bulma, utilizando o formato camel case.

**Figura 24** - Código conjunto de botões BLE

```
import React from 'react'
import { Buttons, Button } from '@brightleaf/elements'

export default () => {
  return (
    <Buttons>
      <Button isPrimary>Primary</Button>
      <Button isInfo>Info</Button>
      <Button isDanger>Danger</Button>
      <Button isLink>Link</Button>
      <Button isWarning>Warning</Button>
      <Button isSuccess>Success</Button>
    </Buttons>
  )
}
```

**Fonte**: https://brightleaf.dev/elements/#/buttons

**Figura 25** - Resultado gráfico conjunto de botões BLE




**Fonte**: https://brightleaf.dev/elements/#/buttons

As *props* isPrimary, isInfo, isDanger, isLink, isWarning e isSuccess do exemplo fornecido na **Figura 24**, representam nomes de cores do Bulma, essas props serão mapeadas internamente pelo componente Button e retornarão para o componente o nome da classe do Bulma a qual elas representam. No exemplo fornecido isPrimary representa a classe “is-primary”, isInfo a classe “is-info” e as outras seguem o mesmo padrão.

#### 3.1.1.1.2 Utilização de componentes complexos

Para a criação de componentes mais complexos a solução apresenta uma característica de grande particionamento do “pedaço” de interface que será exibido ao usuário. Isto é, para criar um determinado componente, como uma Barra de Navegação, é necessário informar vários outros componentes React que representam partes da barra de navegação.

Para criar uma simples barra de navegação utilizando a solução, se faz necessário informar um grande número de componentes. Dois pontos negativos podem ser notados com essa abordagem, a necessidade de aprender como utilizar o componente de forma individual e em grupo e a grande replicação de tags, de código, mesmo utilizando uma solução que busca reduzir a quantidade de código e o esforço de desenvolvimento.

**Figura 26** - Código barra de navegação BLE

```
/>
<NavBarMenu id="navbarBasicExample">
  <NavBarStart>
    <NavBarItem><a>Home</a></NavBarItem>
    <NavBarItem><a>Documentation</a></NavBarItem>
    <NavBarDropDown title="Examples">
      <NavBarItem><a to="/containers">Container</a></NavBarItem>
      <NavBarItem>
        <a to="/columns">Columns</a>
      </NavBarItem>
      <NavBarItem>
        <a to="/notifications">Notifications</a>
      </NavBarItem>
      <NavBarItem>
        <a to="/autocomplete">AutoComplete</a>
      </NavBarItem>
      <NavBarDivider />
      <NavBarItem>
        <a>Report an issue</a>
      </NavBarItem>
    </NavBarDropDown>
  </NavBarStart>
```

**Fonte**: https://brightleaf.dev/elements/#/navbars

**Figura 27** - Resultado gráfico barra de navegação BLE


**Fonte**: https://brightleaf.dev/elements/#/navbars

Como vantagem, o grande particionamento fornecido pela solução, oferece também uma grande customização, é possível executar diversas operações em cada pequena parte do componente e alterar de maneira bastante flexível essas partes.

Um exemplo onde esse particionamento aumenta de maneira considerável o código envolvido na invocação de um componente é ao utilizar o elemento Table. Na **Figura 28** é possível notar que para informar as linhas do cabeçalho da tabela e do corpo é necessário informar um componente TableCell ou um componente TableHeaderCell. Além da necessidade de se lembrar dessa nomenclatura, é preciso também repetir as tags para adicionar novas linhas, essa repetição ao construir uma

pequena tabela não trará muitos problemas ao desenvolvedor, mas em casos de aplicações onde a tabela é mapeada de maneira dinâmica, se tornará necessário a criação de várias linhas HTML, cabeçalhos e rodapés manualmente, usando duas tags para cada unidade, acarretando em muito código.

**Figura 28** - Código Tabela BLE

```
import React from 'react'
import { Table, TableHead, TableRow, TableHeaderCell, TableBo

export default () => {
  return (
    <Table isBordered>
      <TableHead>
        <TableRow>
          <TableHeaderCell>One</TableHeaderCell>
          <TableHeaderCell>Two</TableHeaderCell>
        </TableRow>
      </TableHead>
      <TableBody>
        <TableRow>
          <TableCell>Three</TableCell>
          <TableCell>Four</TableCell>
        </TableRow>
      </TableBody>
    </Table>
  )
}
```

**Fonte**: https://brightleaf.dev/elements/#/tables

**Figura 29** - Resultado Gráfico Table BLE

| One | Two |
|---|---|
| Three | Four |

**Fonte**: https://brightleaf.dev/elements/#/tables

### *3.1.1.2 React Bulma Components*

O pacote *React Bulma Components* é um conjunto de componentes utilizados para auxiliar no desenvolvimento React e Bulma. Basicamente possui os mesmos objetivos que o trabalho correlato detalhado na seção 3.1.1.1, o Brightleaf Elements, e foi criado por um usuário do GitHub chamado apenas de John (sem mais informações obtidas). Possui uma característica interessante, foi o pacote mais retornado nos resultados da pesquisa no *Google*, as referências em quase todas situações apontavam para esse pacote. Observa-se que a solução existente foi observada conforme consta na sua versão 3.2.0, com o commit anterior em 03/03/2020, assim como sua documentação presente em https://couds.github.io/react-bulma-components.

#### 3.1.1.2.1 Propriedades estéticas

Para modificar atributos estéticos utilizando a presente solução correlata, é necessário informar ao componente *React* o nome do atributo e o novo valor que ele receberá. Os nomes dos atributos passados aos componentes não seguem o padrão do Bulma, foram criados da forma como o autor da solução julgou adequada.

**Figura 30** - Código botão RBC

```
<Button
  fullwidth={boolean('Full width', false)}
  color={select('Color', colors)}
  loading={boolean('Loading', false)}
  outlined={boolean('Outlined', false)}
  inverted={boolean('Inverted', false)}
  disabled={boolean('Disabled', false)}
  text={boolean('Text', false)}
  remove={boolean('Remove', false)}
  isStatic={boolean('Static', false)}
  rounded={boolean('Rounded', false)}
  onClick={action('Button Click')}
  onMouseEnter={action('Hover')}
>
  Button
</Button>
```

**Fonte**: https://couds.github.io/react-bulma-components/?path=/story/button--default.

3.1.1.2.2 Utilização de componentes complexos

Para a construção de componentes com um nível de complexidade maior, a presente solução correlata utiliza, assim como a *Brightlef Elements*, um grande particionamento como forma de construir o componente.

**Figura 31** -Código barra de navegação RBC

```
const Navbar= () => (<div className="columns"><div className="column is-half">
    <Navbar color={'primary'} active={false} transparent={false}>
  <Navbar.Brand>
    <Navbar.Item renderAs="a" href="#">
      <img src="https://bulma.io/images/bulma-logo.png" alt="Bulma: a modern CSS
    </Navbar.Item>
    <Navbar.Burger />
  </Navbar.Brand>
  <Navbar.Menu >
    <Navbar.Container>
      <Navbar.Item dropdown hoverable href="#">
        <Navbar.Link arrowless={false}>First</Navbar.Link>
        <Navbar.Dropdown>
          <Navbar.Item href="#">
            Subitem 1
          </Navbar.Item>
          <Navbar.Item href="#">
            Subitem 2
          </Navbar.Item>
        </Navbar.Dropdown>
      </Navbar.Item>
      <Navbar.Item href="#">
        Second
      </Navbar.Item>
    </Navbar.Container>
    <Navbar.Container position="end">
      <Navbar.Item href="#">
            At the end
      </Navbar.Item>
    </Navbar.Container>
  </Navbar.Menu>
</Navbar>
```

**Fonte**: Adaptado de https://couds.github.io/react-bulma-components/?path=/story/navbar--default

**Figura 32** - Resultado gráfico Barra de Navegação RBC


**Fonte**: Próprio Autor,2020

Os pontos negativos da presente solução na questão do grande particionamento são iguais aos da solução *Brightleaf*, pois possui uma grande capacidade de customização, mas ao mesmo tempo exige uma verbosidade muito grande, um considerável número de componentes para construção de outro componente e dificuldades de memorização de cada item, além de uma "fuga" muito grande da nomenclatura do Bulma.

Um ponto negativo adicional que a presente solução correlata possui está na nomenclatura dos atributos estéticos, que como antes mencionados, não estão relacionados com os nomes das classes do Bulma, necessitando de um esforço adicional do usuário em decorar a forma como o autor resolveu nomear esses atributos. Outro ponto negativo encontrado nessa abordagem, é a necessidade de além de informar o nome do atributo, informar em várias situações o valor dele, sendo necessário em situações booleanas, declarar se um atributo é verdadeiro ou falso, algo que se repetirá incessantemente pelo código, pois grande parte das características que serão adicionadas a um componente, possuem uma natureza binária, demandando a informação atributo/valor em diversas ocasiões.

O ponto positivo da biblioteca, como antes mencionado, é o do alto poder de customização dos componentes gerado pelo grande particionamento. Outro ponto positivo que vale ressaltar, é que a presente solução correlata fornece a possibilidade de alteração do CSS do Bulma, isto é, permite que as classes fornecidas pelo Bulma sejam alteradas (via SASS), não usando o "default" encontrado no arquivo .css que representa o resultado final do *framework* Bulma, essa característica adiciona maiores possibilidades de customização estéticas aos desenvolvedores.

# 4 DESENVOLVIMENTO

## 4.1 SOLUÇÃO PROPOSTA

No presente tópico é apresentada a solução desenvolvida para alcançar os objetivos definidos.

Para definir o que seria criado com a solução, diversos requisitos funcionais e não funcionais foram elicitados. Todos os requisitos foram criados após profunda análise teórica e prática sobre os componentes que possibilitaram o alinhamento do requisito a qualidade da solução e ao tempo definido para sua criação. Os requisitos da solução proposta são expressos na **Tabela 2** e **Tabela 3**.

**Tabela 2** - Requisitos funcionais

| **Requisitos Funcionais** | |
|---|---|
| RF | Descrição |
| 1 | A solução deve implementar o conceito de interação |
| 2 | Um componente React deve ser criado para cada elemento julgado importante para a solução |
| 3 | Um arquivo .flow deve ser gerado para cada componente React da solução |
| 4 | A solução deve possibilitar averiguação de tipos em tempo de compilação aos usuários |
| 5 | A solução deve impedir que dados de tipos errados sejam usados pelos componentes |

**Fonte**: O próprio autor,2020

**Tabela 3** - Requisitos não funcionais

| **Requisitos Não Funcionais** | |
|---|---|
| RNF | Descrição |
| 1 | A nomenclatura da parte estética da solução deve ser igual a fornecida pelo Framework Bulma |
| 2 | Os componentes React da solução precisam ser todos do tipo *stateless* |
| 3 | Os componentes React da solução devem ser criados através de um modelo pré-definido |
| 4 | Os atributos "props" dos componentes React da solução devem ser padronizados |
| 5 | A solução deve funcionar com o framework Bulma na versão 0.9.0 |
| 6 | A solução deve funcionar com a biblioteca React na versão 16.13.1 |
| 7 | A solução deve ser disponibilizada no repositório NPM |
| 8 | A solução deve ter uma documentação em formato Github Wiki disponibilizada |

**Fonte**: O próprio autor,2020

Faz-se necessário pontuar que a solução proposta é construída utilizando componentes funcionais criados a partir da ideia de programação funcional definida em 2.4.

A solução proposta busca gerar um resultado equilibrado entre customização e facilidade no uso, optando por adicionar uma certa customização aos componentes sem adicionar complexidades desnecessárias.

Manteve-se a nomenclatura do Bulma, principais motivos: não precisar aprender uma nova nomenclatura, permitir a compatibilidade com os elementos fornecidos pelo Bulma e maior flexibilidade ao definir os elementos.

Os componentes desenvolvidos na presente solução abrangem os grupos Columns, Layout, Form, Elements e Components. Observa-se que existem adaptações quanto ao conteúdo proposto na documentação do Bulma e a implementação na presente solução

### 4.1.1 Estrutura Geral

No presente tópico são apresentados os aspectos fundamentais da presente solução.

De todas as características brevemente explicadas anteriormente, duas ideias são fundamentais para compreensão da presente solução e de como ela busca diminuir o código gerado na integração React/Bulma, que são a criação de uma estrutura pré-definida para os componentes e a interação para criação de código HTML.

#### *4.1.1.1 Estrutura pré-definida*

A ideia dessa característica é de criar o componente desejado a partir de um modelo e com isso diminuir uma parte da codificação necessária. Os modelos utilizados para confecção da maioria dos componentes foram os disponibilizados pela documentação do Bulma, onde eles foram analisados e programados através do React de uma forma que a codificação por parte do usuário da solução proposta fosse menor do que a codificação original.

É possível exemplificar essa estrutura através do elemento DropDown do Bulma. Na **Figura *33*** é possível ver que o Bulma fornece uma estrutura para um tipo de DropDown que pode ser criado (na documentação vários exemplos são fornecidos), a partir desse e de outros exemplos foi implementado o componente DropDown da presente solução, que busca oferecer uma forma de replicar todos os exemplos fornecidos pelos Bulma só que exigindo menos código por parte do programador.

**Figura 33** - Elemento DropDown original Bulma

```
<div class="dropdown is-active">
  <div class="dropdown-trigger">
    <button class="button" aria-haspopup="true"
aria-controls="dropdown-menu">
      <span>Dropdown button</span>
      <span class="icon is-small">
        <i class="fas fa-angle-down" aria-
hidden="true"></i>
      </span>
    </button>
  </div>
  <div class="dropdown-menu" id="dropdown-menu"
role="menu">
    <div class="dropdown-content">
      <a href="#" class="dropdown-item">
        Dropdown item
      </a>
      <a class="dropdown-item">
        Other dropdown item
      </a>
      <a href="#" class="dropdown-item is-active">
        Active dropdown item
      </a>
      <a href="#" class="dropdown-item">
        Other dropdown item
      </a>
      <hr class="dropdown-divider">
      <a href="#" class="dropdown-item">
        With a divider
      </a>
    </div>
  </div>
</div>
```

**Fonte**: https://bulma.io/documentation/components/dropdown/

Utilizando o exemplo do modelo da **Figura *33*** e modificando ele, foi possível chegar a implementação do componente DropDown da presente solução, que uma parte do código é apresentada na **Figura 34**. Na **Figura 34** é possível observar 4 seletores do Bulma predefinidos na implementação do componente da presente solução. Com a predefinição de dropdown-trigger, dropdown-menu e dropdown-content é possível eliminar a necessidade do uso de 6 tags ao se criar um DropDown com o modelo utilizado, mostrando que só de pré-definir uma estrutura, a redução de código já pode ocorrer.

Esse método de redução foi usado no desenvolvimento de todos os componentes, sempre que uma implementação começava era necessário pegar um

modelo e refiná-lo até gerar um componente. Observa-se que em parte dos casos utilizar da estrutura pré-definida já gerava um melhor reuso

**Figura 34** - Trecho do código do componente DropDown

```
18  const DropDown = (props: Props) => {
19      validate(props, props_obj,"DropDown");
20      let definition = (props.definition == undefined) ? "dropdown" : "dropdown " + props.definition;
21      return (<div className={definition}>
22          <div class="dropdown-trigger">
23              {extractButton(props.itens, props.onClick)}
24          </div>
25          <div class="dropdown-menu">
26              <div class="dropdown-content">
27                  {assemble(props.itens)}
28              </div>
29          </div>
30      </div>
31      );
32  }
```

**Fonte**: O próprio autor,2020

A implementação do conceito de estrutura pré-definida está cumprindo o requisito não funcional 3 (RNF3) conforme descrito na **Tabela 3**.

#### *4.1.1.2 Interação para criação de código HTML*

Outra característica da solução que contribui com o reuso de software é o da interação para criação de código HTML.

Alguns componentes da presente solução como Tabs utilizam de itens para construir o seu conteúdo. O componente Tabs possui itens que representam as suas abas, que se construídas através do uso convencional do React e Bulma, faz-se necessário digitar duas tags para cada unidade ou criar métodos para a montagem do HTML dinamicamente. Na **Figura 35** é possível notar que da linha 20 até a 27 é criado o elemento Tabs através da forma convencional e na linha 30 através da solução proposta.

Na linha 30 da figura, é possível observar que itens (linha 9 até 13) é passado como *props* e apresenta os nomes de cada aba especificado no atributo *value* de cada objeto atribuído em itens.

Esse exemplo reflete a propriedade de interação presente na solução proposta onde optar por utilizar código JavaScript (linha 9 até 13) diminui a digitação de tags

HTML e elimina a necessidade de métodos especiais que criam código HTML para cada unidade de um componente.

**Figura 35** - Exemplo Tabs

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.css';
import Tabs from './components/Tabs'
import Column from './components/Column';
import Columns from './components/Columns';


var itens = [{ value: "Pictures", definition: "is-active" },
{ value: "Musica" },
{ value: "Videos" },
{ value: "Documents" }
];

ReactDOM.render(
    <Columns>
        <Column >
            <div className="container is-fluid">

             <div class="tabs">
                    <ul>
                        <li class="is-active"><a>Pictures</a></li>
                        <li><a>Music</a></li>
                        <li><a>Videos</a></li>
                        <li><a>Documents</a></li>
                    </ul>
                </div>
            </div>

            <Tabs itens={itens} />

        </Column>
    </Columns>

    ,
    document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

A implementação do conceito de interação para criação de código HTML cumpre com o requisito funcional 1 (RF1), definido na **Tabela 2**.

### 4.1.2 Comportamento Geral

No presente tópico é apresentada a parte comportamental dos componentes. Por existir uma variabilidade do funcionamento de cada componente, optou-se por selecionar alguns componentes e detalhar seu comportamento. O tópico está divido em componentes independentes e componentes dependentes.

#### *4.1.2.1 Componentes independentes*

No presente subtópico são apresentados o comportamento genérico dos componentes que não dependem de outros componentes da solução para o seu funcionamento, mostrando o fluxo de execução dos componentes Button e Checkbox.

Faz-se necessário pontuar uma característica excepcional nos diagramas de sequência que serão apresentados no presente tópico, os *lifetime* não se referem a classe conforme encontrada na orientação a objetos, mas sim tratam de arquivos JS que não possuem a mesma semântica das classes do paradigma de orientação a objetos, pois conforme explicado em 4.1, trabalha-se na presente solução com uma abordagem funcional.

**Figura 36** - Código Button ARB

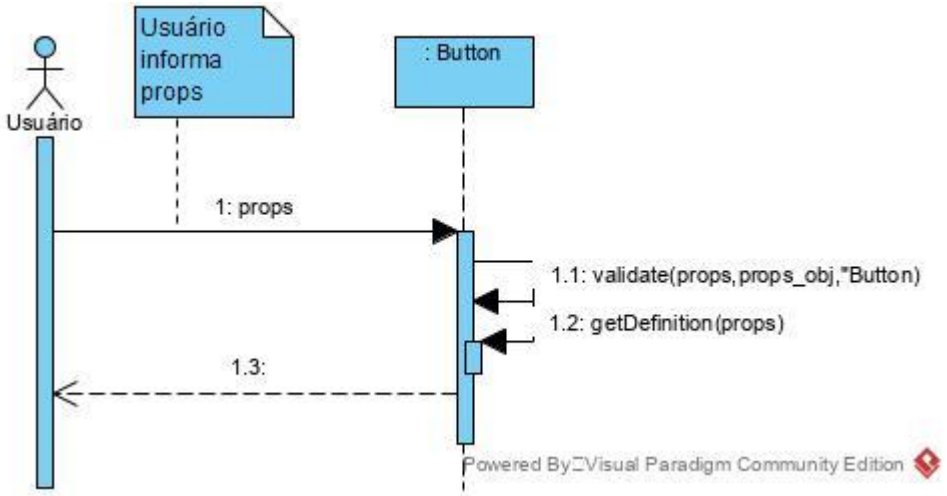
```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.min.css'
import {Button} from 'assemble-react-bulma'

ReactDOM.render(<Button definition="is-danger" label="Botão"/>,document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

**Figura 37** - Resultado Gráfico Button ARB


**Fonte**: O próprio autor,2020

**Figura 38** - Comportamento do componente Button.




**Fonte**: O próprio autor, 2020

Conforme é observado na **Figura 38**, no início do diagrama é possível notar que o usuário informa as *props* para o componente Button (mensagem 1:props), ou seja, fornece todas as informações (parâmetros) necessárias para que Button funcione conforme o esperado, no caso do exemplo a informação passada pelo usuário é uma string “is-danger” (definition) e uma string “Botão”(label) através da *props* definition (conforme mostrado na **Figura 36**).

A passagem de *props*, inicia o processo de montagem do componente, iniciando a execução de uma função, essa função é o componente Button propriamente dito, (conforme explicado no tópico 4.1), logo, a maior barra de execução existente no *lifetime* de Button é referente a toda execução do próprio componente. Para fins de explicação, essa barra será chamada de “barra de execução button”. O objetivo da barra de execução de button é criar um trecho de código HTML que representa um botão HTML com as características definidas pelo Bulma resultando em um componente React.

Após o início da execução representada em barra de execução button, a função validate é invocada e em seguida a função getDefinition é invocada e recebe como parâmetro a *props* informada pelo usuário. A propriedade definition é um atributo de *props* onde o usuário informará propriedade(s) estética(s) que esteja(m) definida(s) pelo Bulma. O objetivo do método getDefinition em Button é criar um *className* a partir de definition para que seja atribuída a tag <button>.

Depois da execução de getDefinition é criado o retorno da função representada por “barra de execução button”, que é um trecho de código HTML. Observa-se que a mensagem 1.3 encontrada na **Figura 38** representa o retorno da barra de execução de button para o usuário.

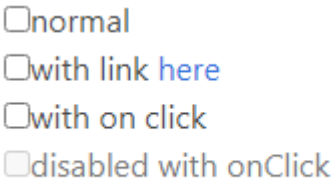
Para complementar o entendimento da parte comportamental da solução é possível observar a dinâmica de outro componente, o CheckBox.

**Figura 39** - Código exemplo CheckBox ARB

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.css';
import { Button, File, Hero, CheckBox, Control } from 'assemble-react-bulma'

ReactDOM.render(
    <div className="container is-fluid">
        <Control>
            <CheckBox>
                normal
            </CheckBox>
        </Control>

        <Control>
            <CheckBox>
                with link <a href="#">here</a>
            </CheckBox>
        </Control>

        <Control>
            <CheckBox onClick={()=>{alert("Clicou CheckBox")}}>
                with on click
            </CheckBox>
        </Control>

        <Control>
            <CheckBox disabled={true} onClick={()=>{alert("Clicou CheckBox")}}>
                disabled with onClick
            </CheckBox>
        </Control>

    </div> ,document.getElementById('root'));
```

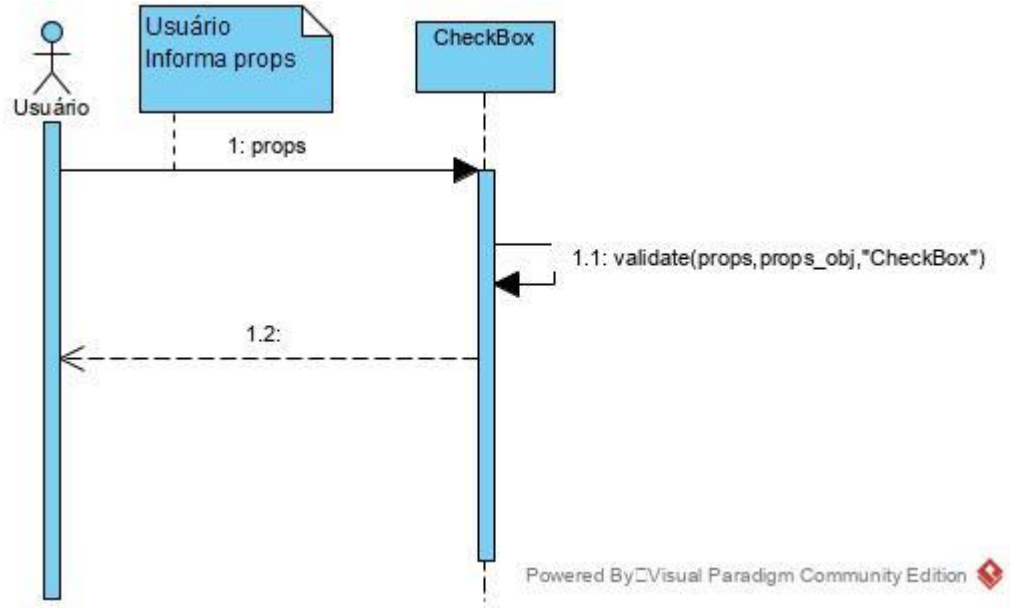
**Fonte**: O próprio autor,2020

**Figura 40** - Resultado gráfico CheckBox ARB




**Fonte**: O próprio autor,2020

**Figura 41** - Comportamento componente CheckBox




**Fonte**: O próprio autor, 2020

Observa-se uma semelhança do comportamento de CheckBox com o comportamento de Button apresentado na **Figura 38**, diferenciando CheckBox de Button apenas em uma chamada adicional de função que Button realiza (getDefinition).

Observando-se os exemplos fornecidos de Button e CheckBox é possível notar que a maior semelhança entre os dois na parte comportamental é: O usuário informa *props*, o componente começa seu processo de montagem e retorna um trecho de código HTML ao final desse processo. A importância de pontuar essa semelhança está no uso desse comportamento em todos os componentes independentes da solução.

#### *4.1.2.2 Componentes dependentes*

Determinados componentes invocarão funções auxiliares durante o processo de montagem (como é o caso de Button na **Figura 38**) e outros irão simplesmente receber *props* e criar um retorno, entretanto na presente solução existem os componentes que funcionam em conjunto com outros componentes, que dependem

de componentes da presente solução para funcionarem, que podem ser chamados de componentes dependentes

A principal diferença entre componentes dependentes e independentes é que os dependentes invocam outros componentes da presente solução, funcionam em conjunto com eles (composição).

Para exemplificar essa diferença são apresentados na **Figura 42** e **Figura 43** os códigos da implementação dos componentes Button e Field. A principal diferença observável é que Button não possui referências para outros componentes da presente solução, enquanto que Field apresenta invocações do componente Control e Input, o que o torna dependente desses dois componentes e quaisquer alterações em Control e Input, refletirão em Field.

**Figura 42** - Parte do código componente Button ARB

```
const Button = (props: Props) => {
    validate(props, props_obj, "Button");
    var definition = getDefinition(props);
    return <button {...props.custom} disabled={props.disabled === true ? true : false}
                   className={definition} onClick={props.onClick}>
                   {props.label}
           </button>
        ;
}
```

**Fonte**: O próprio autor,2020

**Figura 43** - Parte do código componente Field ARB

```
const Field = (props: Props) => {
    validate(props,props_obj, "Field");
    let help_definition = (props.help_definition == undefined) ? "help": "help "+props.help_definition;
    return <div className="field">
        <label className="label">{props.label}</label>
        <Control definition={props.control_definition == undefined ? "has-icons-left" : props.control_definition}>
            <Input  definition={props.input_definition}
                    onChangeEvent={props.onChange}
                    placeholder={props.placeholder}
                    type={props.type}
                    value={props.value}
                    custom={props.custom}
                    ></Input>
            {assembleIcon(props)}
            {props.help_value != undefined &&
            <p className={help_definition}>{props.help_value}</p>
            }
        </Control>
    </div>
}
```

**Fonte**: O próprio autor,2020

### 4.1.3 Estado dos componentes

Conforme definido no tópico 2.9.4, existem duas classificações para os componentes, que são *stateless* e *statefull*. No presente trabalho decidiu-se implementar todos os componentes na forma *stateless*.

Essa decisão está calcada na ideia de *Single Source of Truth* (SSOT), em português “Fonte única da verdade”. Uma tecnologia muito utilizada que implementa o SSOT é o *Redux*, amplamente usada para o gerenciamento de estados ao se desenvolver componentes *React*, principalmente os com complexidade consideravelmente elevada. O SSOT no *Redux* é observado no gerenciamento dos estados, ele provê uma única fonte de dados (estado) em uma árvore de componentes, e quaisquer alterações realizadas nesses dados, são feitas através de referências, e somente essas referências poderão alterar o estado, sendo essa uma característica de “proteção” dos dados e desacoplamento, fornecendo aos componentes maior independência quanto aos valores que manipulará (REDUX,2020).

A motivação de implementar os componentes como *stateless* está na possibilidade de delegar ao usuário a tarefa de gerenciar o estado, retirando dos componentes da solução qualquer responsabilidade que não for reagir a eventos dos usuários e renderizar as mudanças conforme a sua implementação.

### 4.1.4 Padrões de Nomenclatura

O presente tópico trata da explicação dos padrões adotados pela presente solução, que foram desenvolvidos com objetivo de facilitar o aprendizado da solução.

#### *4.1.4.1 Atributos*

O presente subtópico visa mostrar a nomenclatura dos atributos utilizados nos componentes. Primeiramente é explicado um padrão para a construção e uso da nomenclatura da solução e em seguida são descritas as *props* e as variáveis Javascript utilizadas.

Existe uma divisão de conceito que precisa ser pontuada, toda manipulação de característica que pode ser atingida através do Bulma (adicionando nomes de classes do Bulma no atributo “class”), é possibilitada pelo uso de *definition*, porém existem alterações que são resultantes da manipulação de outros atributos do HTML, essas são atingidas por *props* específicas, ambos os casos são detalhados a seguir.

**Figura 44** - Chamada do componente Input ARB

```
<Input definition="is-loading" type="password" placeholder="input"/>
```

**Fonte:** O próprio autor,2020

**Figura 45** - Código gerado pelo componente Input ARB

```
<div class="control is-loading">
  <div class="control">
    <input type="password" class="input is-loading" placeholder="input" />
  </div>
</div>
```

**Fonte**: O próprio autor,2020

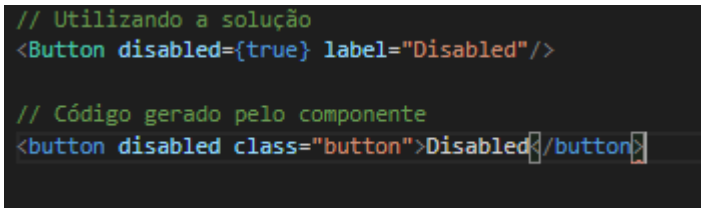
**Figura 46** - Resultado gráfico do componente Input ARB.


**Fonte**: O próprio autor, 2020

Observa-se que na Figura 45 o código resultante mostra a sentença “control is-loading”, que é uma sentença válida que gera um resultado estético formado por duas sentenças do Bulma (control e is-loading). Neste caso, a propriedade do Bulma “is-loading” é definida através do atributo className da primeira tag  do exemplo, sendo assim, o componente Input da presente solução, recebe essa propriedade através da props definition, conforme observado na Figura 44.

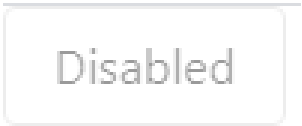
Na **Figura 47** é apresentada a chamada do componente Button da presente solução e o código gerado, na **Figura 48** é apresentado o resultado gráfico do componente.

**Figura 47** - Componente Button e código resultado




**Fonte**: O próprio autor,2020

**Figura 48** - Resultado Gráfico Button Disabled




**Fonte**: O próprio autor,2020

No exemplo fornecido na **Figura 47** é possível observar que no código gerado pelo componente existe um atributo *disabled*, isto é, uma tag diferente de *class* que está alterando a estética da tag <button> e isso é refletido no componente Button da presente solução, que não possui essa estética especificada no definition, mas sim uma props especial para trata-la, no caso “disabled={true}”.

Conforme definido em 4.1.2.1 e 4.1.2.2 a presente solução possui componentes dependentes e independentes. Ao utilizar um componente dependente, em diversos casos, é possível passar valores para os componentes das quais ele depende e existe um padrão para nomenclatura desse envio de valores. Para exemplificar, será utilizado o exemplo do componente Field. Na **Figura 49** é possível notar que o componente Field invoca Control e Input da presente solução e passa para esses componentes *definition* como *props*, entretanto no componente Field esses valores informados são *props.control_definition* (para o definition de Control) e *props.input_definition* (para o definition de Input). O padrão exemplificado é constituído pelo nome do componente filho que o usuário deseja modificar (no caso do exemplo input e control) e o parâmetro que está sendo alterado. Na **Figura 49** ainda é possível notar que alguns parâmetros informados para componente filho de Field não possuem o padrão antes mencionado, isso pois, *definition* está presente em dois componentes (Control e Input), logo foi necessário distinguir as propriedades

através do nome, ao contrário de placeholder e type por exemplo, que somente o componente Input possui.

**Figura 49** - Exemplo nomenclatura componente dependente

```
const Field = (props: Props) => {
    validate(props,props_obj, "Field");
    return <div className="field">
        <label className="label">{props.label}</label>
        <Control definition={props.control_definition == undefined ? "has-icons-left" : props.control_definition}>
            <Input  definition={props.input_definition}
                    onChangeEvent={props.onChange}
                    placeholder={props.placeholder}
                    type={props.type}
                    value={props.value}></Input>
            {assembleIcon(props)}
        </Control>
    </div>
}
```

**Fonte**: O próprio autor, 2020

Os atributos da presente solução serão divididos em dois grupos, os de comportamento e os de aparência.

#### 4.1.4.1.1 Aparência

Nas tabelas da presente seção são descritos os atributos dos componentes React que tratam a aparência desse componente, ou seja, dos seletores CSS predefinidos pelo Bulma que são adicionados nas tags HTML de cada componente.

**Tabela 4** – Tabela de atributos I

| **Atributo** | **Descrição** |
|---|---|
| body_definition | Atributo para definir o estilo de um Button |
| button_definition | Estilo do definition de body do componente Hero |
| card_header_definition | Atributo para definer estilo de um cabeçalho de Card |
| card_image | Imagem do componente Card |
| card_content | Conteúdo do componente Card |
| card_image_definition | Atributo para definir o estilo da imagem de Card |
| card_head_name | Header do componente Card |
| card_header_definition | Atributo para definer estilo de um cabeçalho de Card |
| children | |
| container_definition | Atributo para definir estilo de um container |
| content_definition | Atributo que define o estilo de um componente Content |
| | |
| control_definition | Atributo que define o estilo de um componente Control |
| custom | Atributo para uso do JSX Spread. |
| custom_header(card) | Cabeçalho customizado |
| definition | Em inglês “definição”, é uma variável que recebe as características estéticas do componente React representada em forma de string. É nela que o usuário da aplicação informa as características do framework Bulma que ele deseja que apareça visualmente. A propriedade definition quando informada através de props representa as características do componente e quando enviada através de array simboliza as características do item |
| delete | Usado para informar que o componente possui um botão com o ícone “x” que simboliza “fechar”. |
| disabled | Usado para desativar um componente. |
| field | Usado no componente ButtonList para especificar que ele aceita outros componentes que não sejam Button. |
| field_definition | |
| figure_definition | Atributo que define o estilo da tag figure do componente Image |
| filename | Nome do rótulo do componente File. |
| Foot_definition | Estilo do definition de footer do componente Hero |
| footer(TableQuery) | Atributo binário que define se o rodapé de TableQuery é visível ou não. |

**Fonte**: O próprio autor,2020

**Tabela 5** - Tabela de atributos II

| Atributo | Descrição |
|---|---|
| footer_definition | Atributo para definir estilo de um rodapé |
| head_definition | Estilo do definition de header do componente Hero |
| header (TableQuery) | Atributo binário que define se o cabeçalho de TableQuery é visível ou não. |
| header_definition | Atributo para definir estilo de um caneçalho |
| height_brand | Altura do brand de NavigationBar |
| help_definition | Definition do elemento "help" de Field |
| help_value | Valor do elemento "help" de Field |
| href_next | URL para redirecionamento do link “next” |
| href_previous | URL para redirecionamento do link “previous” |
| icon_definition | Atributo para propriedades estéticas de ícones |
| icon_name | Props que define qual ícone do Font Awesome será utilizado |
| icon_onClick | Evento disparado ao clicar em icon |
| image_definition | Atributo para definir o estilo de uma imagem |
| image_onClick | Evento disparado ao clicar em uma imagem |
| input_definition | Atributo para definir o estilo de Input |
| key | Usado para identificar unicamente um elemento. Usado principalmente em listas. |
| label | Rótulo que o componente possui |
| left_definition | Atributo de estilo de “media-left” do componente MediaObject |
| level_definition | Atributo de estilo do componente Level |
| link | Atributo onde é informado um link para imagem |
| link_brand | Link do brand de NavigationBar |
| loading | Característica onde o componente apresenta um ícone dinâmico de um círculo girando. |
| max | Valor máximo |
| multiple_size | Variar o tamanho de um componente Select |
| name | Name componente Radio |
| next_disabled | Link “next” desabilitado |
| next_name | Nome do link “next” |

**Fonte**: O próprio autor,2020

**Tabela 6** - Tabela de atributos III

| **Atributo** | **Descrição** |
|---|---|
| p | Define se os componentes Title e Subtitle são criados a partir da tag <p> |
| placeholder | Props em componentes que são ou possuem formulários (Field, TextArea, Input e campo de pesquisa de Panel). |
| previous_disabled | Link “previous” desabilitado |
| previous_name | Nome do link “previous’ |
| readonly | Define se o componente é somente leitura |
| ref | |
| ref_data | |
| right | Define o lado direito para alguma parte do componente |
| right_definition | Atributo de estilo de “media-right” do componente MediaObject |
| rows | Props utilizado para manipular tamanho do TextArea |
| search_definition | Atributo de estilo para campo “search” |
| search_icon_definition | Atributo para definir estilo do ícone de “search” |
| search_placeholder | Placeholder para campo “search” |
| spaced | Atributo binário que determina se existe espaçamento em um componente Tile |
| src | Atributo onde é informado a localização de um recurso, como uma URL que acesse uma imagem por exemplo. |
| subtitle | Subtítulo |
| subtitle_definition | Atributo para definir o estilo de um subtítulo |
| table_definition | Atributo para definir o estilo de Table (tag <table>) |
| title | Título de alguma parte pré-definida do componente |
| title_definition | Altera a aparência referente ao título, como cor, tamanho e etc. |
| type | Tipo do texto informado em um formulário |
| value | Em inglês significa valor, logo essa variável é utilizada para expressar valores de diversos tipos, como caracteres, objetos, podendo se estender aos componentes React e etc. Essa variável é muitas vezes encontrada ao se utilizar componentes que possuam uma lista, ou que seja necessário o usuário informar diversos valores que possuam estrutura semelhante. |
| width_brand | Comprimento do brand de NavigationBar |

**Fonte**: O próprio autor,2020

#### 4.1.4.1.2 Comportamento

Na tabela da presente seção, são apresentados os atributos que alteram a parte comportamental dos componentes como os eventos e os itens que o “preencherão”.

**Tabela 7** - Tabela de atributos comportamentais

| Atributo | Descrição |
|---|---|
| itens | É um array de objetos JavaScript que na maior partes dos casos é utilizado para informar valores que farão parte de uma lista de algum componente |
| itens_blocks | Itens do componente Panel |
| itens_body | Serve para diferenciar os itens que pertencem ao “corpo” de um componente |
| itens_end | Itens do fim do componente NavigationBar |
| itens_footer | Serve para diferenciar os itens que pertencem ao rodapé de um componente |
| itens_header | Serve para diferenciar os itens que pertencem ao cabeçalho de um componente. |
| itens_start | Itens do começo do componente NavigationBar |
| itens_tabs | Itens do componente Tabs |
| onChange | Evento onChange em tag HTML |
| onChangeSearch | Evento onChange do componente Search |
| onClick | Evento de onClick em tag HTML |
| onClickDelete | Evento ao clicar no botão "delete" |
| onClickRow | Evento ao clicar em algumas linha do componente Table |

**Fonte**: O próprio autor,2020

Uma descrição mais detalhada dos atributos é fornecida no APÊNDICE B – COMPONENTES DETALHADOS**.**

A implementação dos atributos seguindo a padronização antes descrita cumpre com requisito não funcional 4 (RNF4) definido na **Tabela 3**

#### *4.1.4.2 Atribuições aos componentes*

As atribuições de valores aos componentes são realizadas através das props (ver tópico 2.9.5). Existem vários tipos de dados que podem ser transmitidos aos componentes React, desde dados primitivos até componentes extremamente complexos.

As soluções correlatas *Brightleaf Elements* e *React Bulma Components* resolveram essa questão optando por criar componentes específicos da solução que são passados como filhos e levam os valores desejados. Na **Figura 26** e **Figura 31** é possível observar que para criação de uma barra de navegação utilizando as soluções, um grande número de componentes foi necesário no processo, as soluções criaram uma quantidade considerável de componentes filhos para atribuir valores, o

que necessita de um aprendizado de nomenclatura e uso de todos os componentes envolvidos.

Ao adotar a criação de tais componentes, as soluções correlatas acabam por delegar ao usuário a responsabilidade por: Aprender como utilizar todos os componentes envolvidos na atribuição e criar um componente para cada valor.

A presente solução busca diminuir algumas dessas responsabilidade e deixar que os componentes resolvam parte dela em determinadas ocasiões, adicionando maior facilidade ao uso. Pontua-se que em alguns casos é necessário gerar exceções, optando pela abordagem de criar componentes específicos, igual encontrado nas soluções correlatas.

Observa-se que todas as formas são comuns ao utilizar o React, pois são suportadas pela tecnologia, porém julgou-se necessário especificar sua utilização na presente solução. A explicação das atribuições é dividida em três tipos: valor único, array e children.

1. Valor único: Trata-se da atribuição de um valor individual fornecida pela tecnologia React, valores dos mais variados tipos (string, int, funções, boleanos e etc.).

2. Array: Utilizado na necessidade de informar diversos valores que fazem parte dos componentes. Encontrado em todos os componentes que possuam conteúdo dinâmico, como tabelas, listas, checkbox e etc.

3. Children: Trata-se dos atributos *children* das props, sendo um valor que um componente possui quando o mesmo possui filho. De maneira informal pode-se dizer que um componente possui um *children* quanto entre a sua tag de abertura e fechamento existe algum valor, esse valor é o filho do componente (nesse caso valor pode ser de qualquer tipo). *Children* é implementado pelo React.

A **Figura 50** e **Figura *51*** mostram um exemplo do uso do componente Tabs da presente solução, com ele é possível observar dois tipos de troca de informações, a com o valor único e a com array.

É possível observar no código a props definition passada normalmente com uma string “is-large” (valor único) que remete a uma característica de tamanho

fornecida pelo Bulma e uma *props* itens (array) que fornece ao componente Tabs os valores das suas abas.

**Figura 50** - Envio de dados componente Tabs

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.min.css'
import {Tabs} from 'assemble-react-bulma'

const itens = [{value:"Home",definition :"is-active"},
               {value: "Musica"},
               {value: "Videos"},
               {value: "Documentos"}
              ];

ReactDOM.render(
<div className="columns is-centered">
  <div className="column is-half">
    <Tabs definition="is-large" itens={itens}/>
  </div>
</div>
,document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

**Figura 51** - Resultado gráfico do componente Tabs.


**Fonte**: O próprio autor, 2020

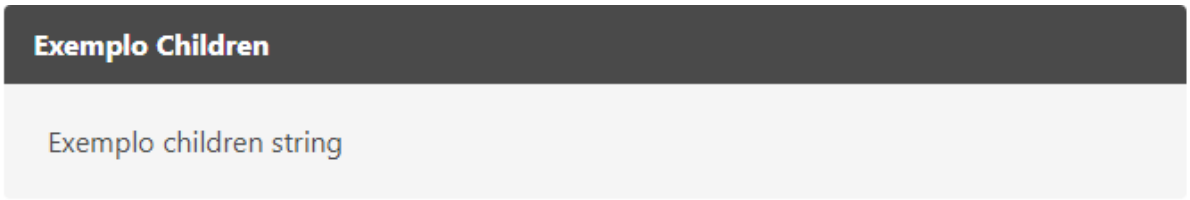
Na **Figura 52** um exemplo de como é passado *children* através da presente solução. Observa-se o valor “Exemplo children string” passado como *children* é uma string, porém poderia ser passado qualquer outro tipo de valor no seu lugar, como é mostrado na **Figura 54** e **Figura 55** onde Message recebe como *children* dois componentes Field.

**Figura 52** - Código componente Message ARB com children string

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.css';
import { Message ,Columns, Column} from 'assemble-react-bulma'

ReactDOM.render(
        <Columns definition="multiline is-centered">
            <Column definition="is-half">
                <Message header="Exemplo Children"> Exemplo children string</Message>
            </Column>
        </Columns>
        ,document.getElementById('root'));
```

**Fonte**: Autor,2020

**Figura 53** - Resultado gráfico Message ARB com children string



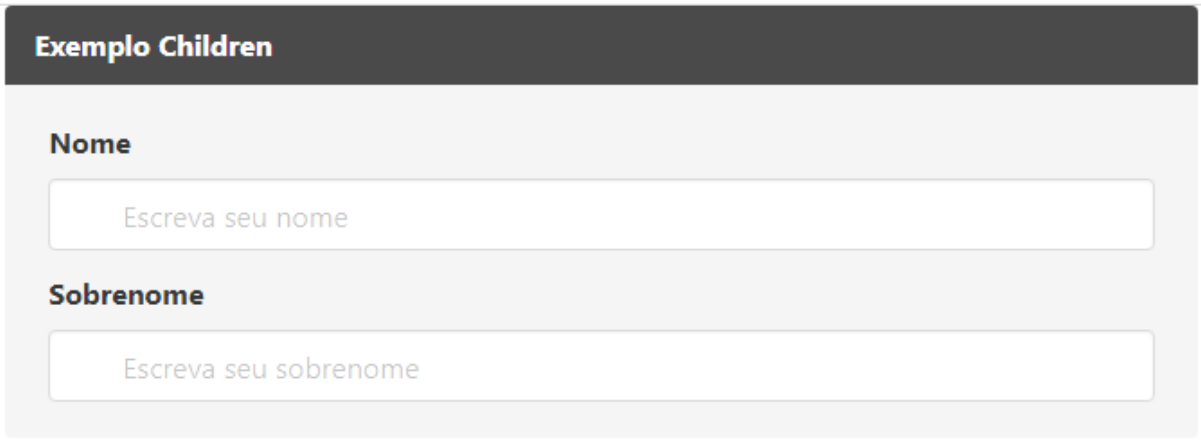

**Fonte**: O próprio autor,2020

**Figura 54** - Código Message ARB com componentes *children*

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.css';
import { Message ,Columns, Column,Field} from 'assemble-react-bulma'

ReactDOM.render(
        <Columns definition="multiline is-centered">
            <Column definition="is-half">
                <Message header="Exemplo Children">
                    <Field label="Nome" placeholder="Escreva seu nome"/>
                    <Field label="Sobrenome" placeholder="Escreva seu sobrenome"/>
                </Message>
            </Column>
        </Columns>
        ,document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

**Figura 55** - Resultado gráfico Message ARB com *childrens* componentes




**Fonte**: O próprio autor,2020

Existe outro tipo de atribuição importante que é a dos eventos do componente. Sendo tratada em detalhes no tópico 4.1.5

### 4.1.5 Eventos

Na solução proposta existem basicamente duas formas de lidar com eventos, de atribuir um evento a um componente. Uma forma é utilizar eventos previamente definidos e suportados pelos componentes e a segunda forma é utilizando o atributo *spread* do JSX.

A ideia de implementar eventos fixos aos componentes visa facilitar o uso dos eventos mais básicos e essenciais ao se criar uma interface gráfica.

Para identificar quais eventos são mais básicos e essenciais, criou-se uma pequena aplicação e durante o desenvolvimento dessa aplicação buscou-se sempre utilizar o mínimo de eventos possível, aqueles indispensáveis para atingir um nível necessário de interação. Com a criação da aplicação foi possível observar que dois eventos são fundamentais para possuir uma interação ínfima em um sistema utilizando *React* e JavaScript, são eles: *onClick* e *onChange*. O evento *onClick* foi necessário em 9 dos 11 componentes React utilizados na aplicação e o evento onChange em 6 componentes. Com esses resultados, sempre que viável, foram implementados esses eventos. Observa-se que classificar um evento como essencial ou até mesmo básico é algo muito subjetivo, a aplicação serviu exatamente para

aproximar essa classificação da realidade, o que não elimina outros eventos de serem essenciais para uma aplicação, pois essa essencialidade pode variar com o contexto.

A segunda forma de atribuir um evento a um componente é utilizando o atributo *spread* do JSX. Essa forma foi fornecida justamente para tentar agregar uma grande possibilidade de eventos utilizáveis ao componente sem a árdua tarefa de os pré-definir e os manter, agregando dinâmica aos eventos e permitindo uma maior customização por partes dos usuários. Observa-se que os desenvolvedores do React recomendam o uso consciente ao utilizar o operador, para que atributos inválidos não sejam enviados ao DOM(REACT,2020).

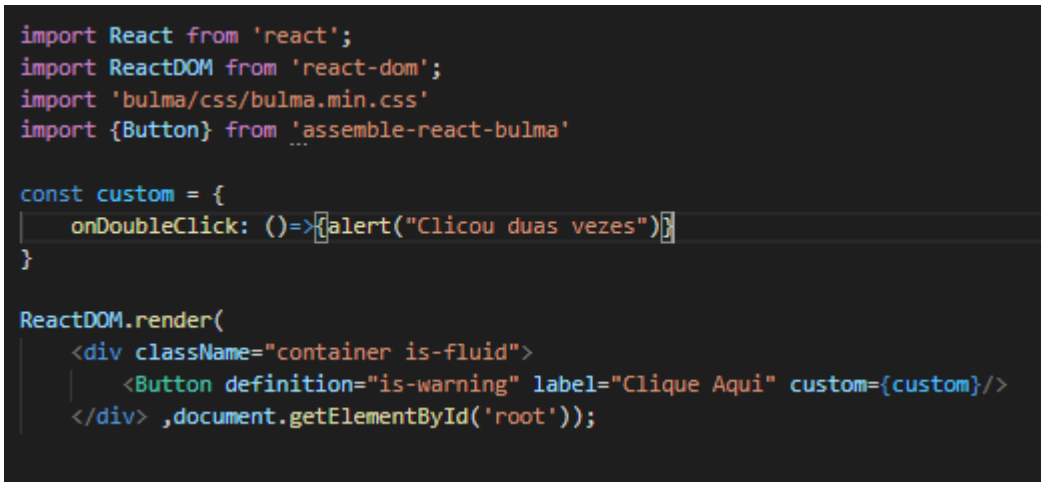
Na **Figura 56** é apresentado o código utilizado o evento *onClick* que é pré-definido pelo componente Button e na **Figura 57** é apresentado o código utilizando o JSX *spread* que informa um evento que não foi pré-definido, no caso o evento *onDoubleClick*.

**Figura 56** - Button onClick pré-definido.

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.min.css'
import {Button} from 'assemble-react-bulma'

const onClick = ()=>{
    alert("Clicou")
}

ReactDOM.render(
    <div className="container is-fluid">
        <Button definition="is-warning" label="Clique Aqui" onClick={onClick}/>
    </div>

,document.getElementById('root'));
```

**Fonte**: O próprio autor, 2020

**Figura 57** - Button Double Click utilizando Spread



```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bulma/css/bulma.min.css'
import {Button} from 'assemble-react-bulma'

const custom = {
    onDoubleClick: ()=>{alert("Clicou duas vezes")}
}

ReactDOM.render(
    <div className="container is-fluid">
        <Button definition="is-warning" label="Clique Aqui" custom={custom}/>
    </div> ,document.getElementById('root'));
```

**Fonte**: O próprio autor,2020

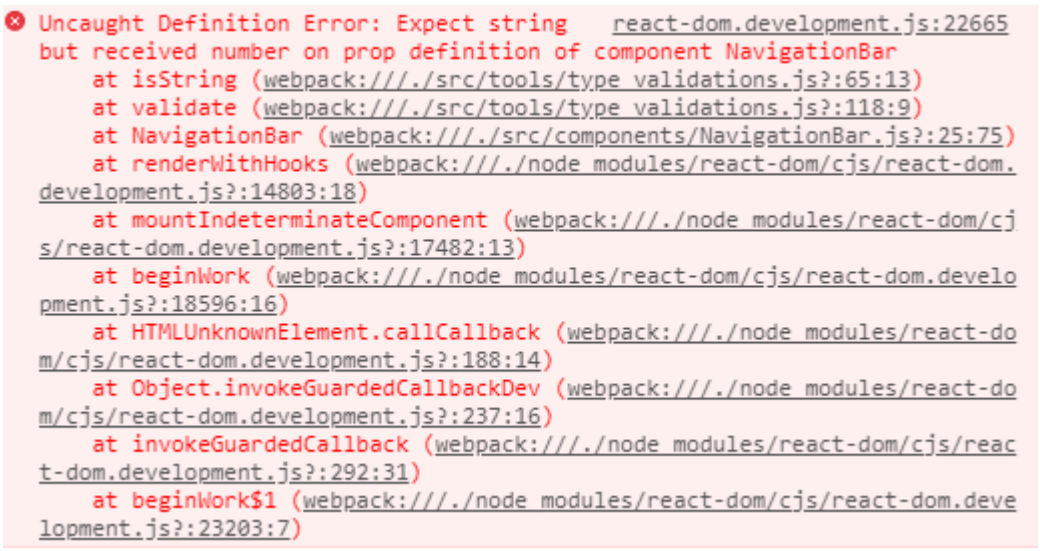
### 4.1.6 Identificação de erros

Na presente solução existem duas estratégias para a identificação de erros: durante o desenvolvimento se o usuário da solução utilizar o Flow ou durante a execução da solução proposta.

O tratamento de erro criado se chama DefinitionError. Esse tratamento serve para identificar erros de tipo não identificados pelo usuário, que são geralmente ocasionados por dois motivos: falta de atenção ao identificar os sinais do Flow(caso usuário utilize Flow) ou caso ele não utilize, evitar que o componente seja utilizado sem os tipos corretos, sem os dados na tipagem adequada, pois conforme definido em 2.4, uma função com parâmetros corretamente tipados apresenta uma melhor qualidade e sendo os componentes da presente solução funcionais, observou-se necessário esse controle.

Caso um tipo informado ao componente não seja o esperado, um erro é lançado com uma mensagem no console do navegador. Um exemplo de mensagem disparada por um erro DefinitionError é exemplificado na **Figura 58**. Nesse caso foi informado ao atributo definition de NavigationBar um valor número, sendo que esse atributo espera um valor string.

**Figura 58** - DefinitionError Mensagem




**Fonte**: O próprio autor,2020

### 4.1.7 Documentação

Para auxiliar no uso da solução, elaborou-se uma documentação no formato *wiki* do GitHub, onde cada componente possui sua respectiva página da *wiki* com exemplos utilizando código e imagens. As páginas wiki também possuem os atributos de cada componente e seu tipo definido pelo Flow.

A ideia de fornecer a documentação nesse formato é possibilitar ao usuário entender como pode utilizar solução para criar suas interfaces gráficas.

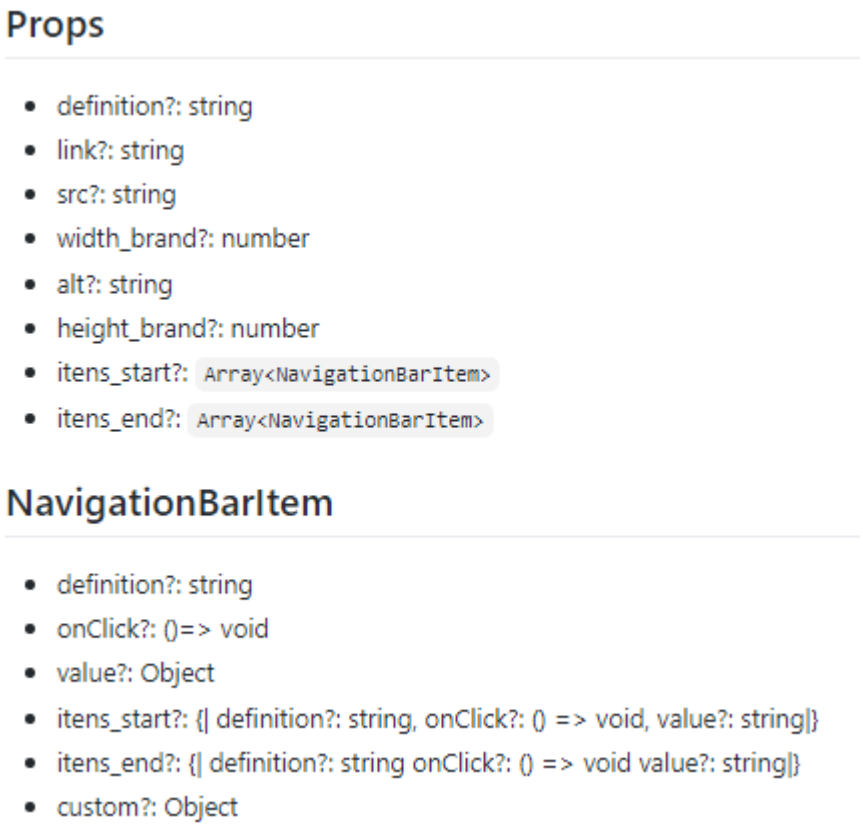
A **Figura 59** demonstra como estão arranjados os exemplos de usos do componente documentado, onde existe um código e um resultado gráfico que corresponde aquele código. Todas as variáveis necessárias para executar os exemplos são fornecidas, no caso desse exemplo, itens_start e itens_end são definidos na mesma página.

Na página wiki de cada componente, também é informado os atributos(*props*) que ele possui. Em casos de componentes que possuam interação (ver seção 4.1.1.2) é informado o tipo de cada objeto da prop itens (no caso da **Figura 60** é NavigationBarItem), conforme mostrado na **Figura 60**.

**Figura 59** - Exemplo documentação Wiki


**Fonte**: O próprio autor,2020

**Figura 60** - Tipo dos objetos de prop Itens




**Fonte**: O próprio autor,2020

### 4.1.8 Testes

Os testes da aplicação foram realizados de duas formas: um arquivo .js com uma série de testes unitários e um arquivo .test.js que o Jest utiliza para averiguar eventuais mudanças indesejadas na renderização dos componentes (JEST,2020).

Os testes unitários foram armazenados em um arquivo .js para cada componente ou para cada componente relacionado (Columns/Column e Button/ButtonList por exemplo). Esses testes unitários tratam de renderizar o componente de uma forma desejada, delegando ao autor analisar se a renderização está como planejada ou se algum erro é lançado.

Os testes com JEST foram úteis durante o desenvolvimento para identificar se alguma mudança na codificação do componente impactava em mudanças do código HTML gerado. Caso contatada alguma mudança no código HTML gerado, o teste unitário do componente era reexecutado para averiguação/alteração.

### 4.1.9 Disponibilidade

A biblioteca utilizável desenvolvida no presente trabalho está localizada no repositório NPM, no link https://www.npmjs.com/package/assemble-react-bulma onde é possível encontrar o comando para usá-la, o link para a documentação completa (wiki), o link para o código fonte e outras informações do pacote desenvolvido.

# 5 RESULTADOS OBTIDOS

O presente capítulo trata dos resultados obtidos com a solução proposta mostrando as métricas, aplicações da solução assim como as melhorias observadas a partir do seu uso prático.

## 5.1 REFATORAÇÃO DE APLICAÇÕES EXISTENTES

Uma das formas de atestar a resolução fornecida pela presente solução foi através do *refactoring* de uma série de aplicações utilizadas na disciplina Programação para Web (INE5646), do curso de Sistemas de Informação da UFSC, no semestre de 2019/2. As aplicações que receberam o *refactoring* são sistemas com baixa complexidade que utilizam a tecnologia React e Bulma para criar suas interfaces gráficas, entretanto não utilizam outras soluções para facilitar o desenvolvimento. Sendo assim, realizou-se o *refactoring* de 3 aplicações e analisou-se a diferença entre os códigos das aplicações originais (construídas sem a solução proposta) e os códigos das aplicações resultantes.

As métricas utilizadas para análise do refactoring são baseadas em linhas e tags. A métrica através da quantidade de linhas possui uma grande subjetividade, pois essa quantidade muitas vezes pode sofrer mudanças pelo modo como o desenvolvedor organiza seu código, portanto optou-se por adicionar também a métrica de quantidade de tags. Observa-se que será utilizada também a métrica de linhas totais, tanto com código HTML quanto com código JavaScript, para a análise do percentual total de redução do código.

O refactoring de cada componente é apresentado na seguinte forma:

1 Uma figura que apresenta o resultado gráfico do componente original
2 Uma figura que apresenta o resultado gráfico do componente após refactoring
3 Uma ou mais figuras que mostram o código completo do componente
4 Uma ou mais figuras que mostram o código completo do componente após o *refactoring*

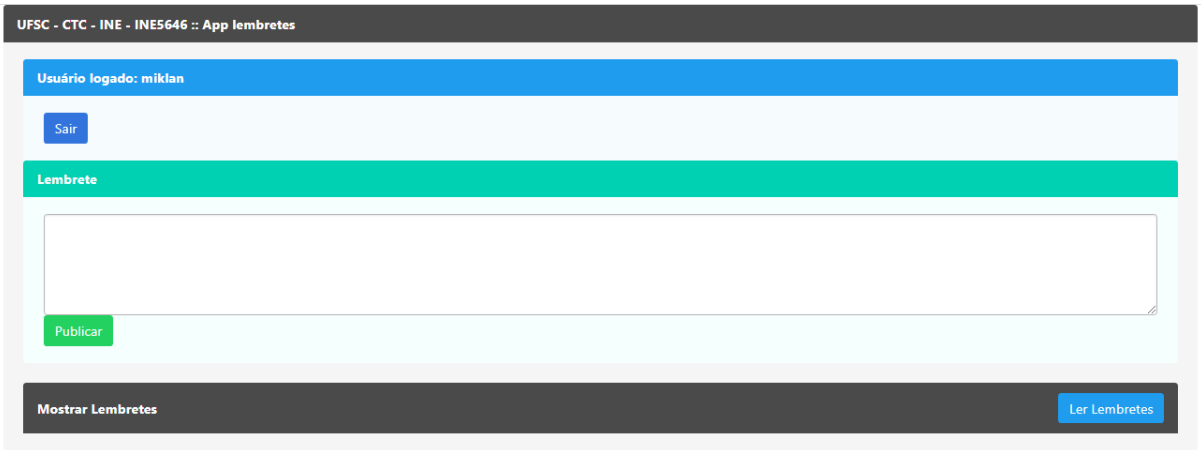
### 5.1.1 App Lembretes

Essa aplicação trata de um pequeno sistema para publicar e ler lembretes, permitindo o cadastro e login de usuários, além da criação, leitura e exclusão de lembretes. A aplicação possui cinco componentes React nos quais será aplicada a presente solução, são eles: App, Login, MostraLembrete, MostraLembretes e PublicaLembretes.

#### *5.1.1.1 App*

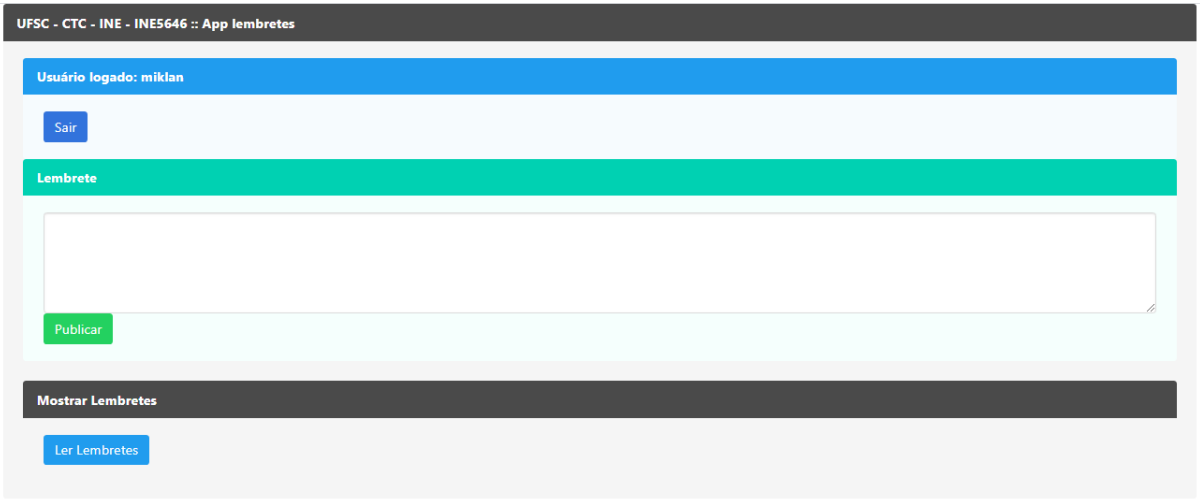
No refactoring de App apenas o componente Message da presente solução foi utilizado.

**Figura 61** - Imagem original do componente App de App Lembretes




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

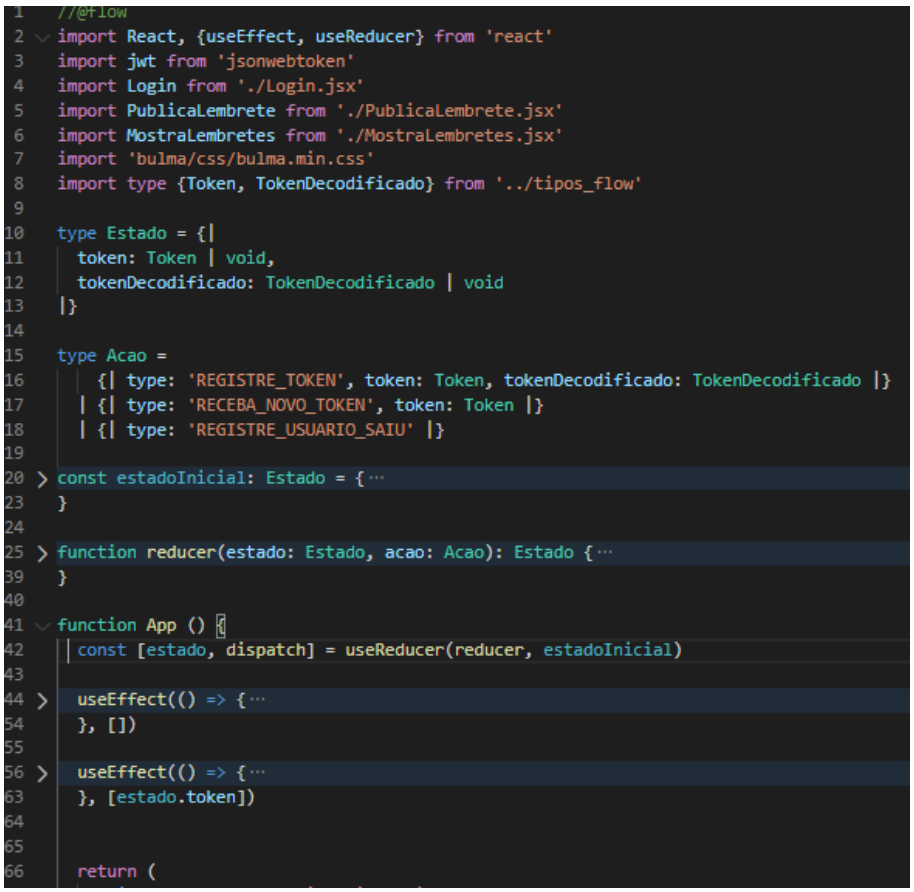
**Figura 62** - Imagem após refactoring do componente App de App Lembrete




**Fonte**: O próprio autor, 2020

Na **Figura 61** e é apresentada a imagem do componente App original e na **Figura 62** é apresentada a imagem do componente App após a aplicação do *refactoring.* A única mudança observada é a realocação do botão “Ler Lembretes” que foi retirado do header do componente Message.

**Figura 63** - Código original App de Lembretes



```
//@flow
import React, {useEffect, useReducer} from 'react'
import jwt from 'jsonwebtoken'
import Login from './Login.jsx'
import PublicaLembrete from './PublicaLembrete.jsx'
import MostraLembretes from './MostraLembretes.jsx'
import 'bulma/css/bulma.min.css'
import type {Token, TokenDecodificado} from '../tipos_flow'

type Estado = {|
  token: Token | void,
  tokenDecodificado: TokenDecodificado | void
|}

type Acao =
    {| type: 'REGISTRE_TOKEN', token: Token, tokenDecodificado: TokenDecodificado |}
  | {| type: 'RECEBA_NOVO_TOKEN', token: Token |}
  | {| type: 'REGISTRE_USUARIO_SAIU' |}

const estadoInicial: Estado = {···
}

function reducer(estado: Estado, acao: Acao): Estado {···
}

function App () {
  const [estado, dispatch] = useReducer(reducer, estadoInicial)

  useEffect(() => {···
  }, [])

  useEffect(() => {···
  }, [estado.token])


  return (
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

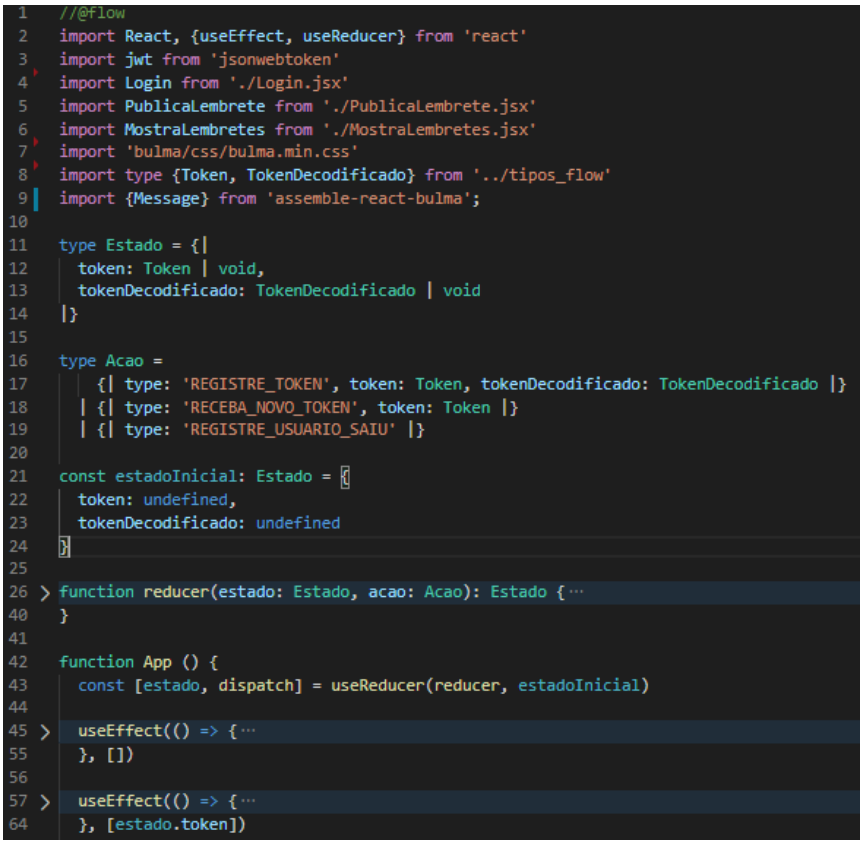
**Figura 64** - Código original App de Lembretes II

```
67      <div className='container is-fluid'>
68        <div className='message'>
69          <div className='message-header'>
70              UFSC - CTC - INE - INE5646 :: App lembretes
71          </div>
72          <div className='message-body'>
73            <Login onToken={token => dispatch({type: 'RECEBA_NOVO_TOKEN', token})}
74              onSaiu={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}
75              token={estado.token}
76              tokenDecodificado={estado.tokenDecodificado}/>
77            {
78              estado.token &&
79                <PublicaLembrete token={estado.token}
80                  onTokenInvalido={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}/>
81            }
82            {
83              estado.token &&
84                <MostraLembretes token={estado.token}
85                  onTokenInvalido={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}/>
86            }
87          </div>
88        </div>
89      </div>
90    )
91  }
92
93  export default App
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

Na **Figura *63*** e **Figura 64** pode-se observar que existem 22 linhas referente ao código HTML (linha 67 até a 89), 11 tags, além do código original possuir um total de 93 linhas.

**Figura 65** - Código refatorado de App de Lembretes I



```
1  //@flow
2  import React, {useEffect, useReducer} from 'react'
3  import jwt from 'jsonwebtoken'
4  import Login from './Login.jsx'
5  import PublicaLembrete from './PublicaLembrete.jsx'
6  import MostraLembretes from './MostraLembretes.jsx'
7  import 'bulma/css/bulma.min.css'
8  import type {Token, TokenDecodificado} from '../tipos_flow'
9  import {Message} from 'assemble-react-bulma';
10
11 type Estado = {|
12   token: Token | void,
13   tokenDecodificado: TokenDecodificado | void
14 |}
15
16 type Acao =
17     {| type: 'REGISTRE_TOKEN', token: Token, tokenDecodificado: TokenDecodificado |}
18   | {| type: 'RECEBA_NOVO_TOKEN', token: Token |}
19   | {| type: 'REGISTRE_USUARIO_SAIU' |}
20
21 const estadoInicial: Estado = {
22   token: undefined,
23   tokenDecodificado: undefined
24 }
25
26 function reducer(estado: Estado, acao: Acao): Estado {···
40 }
41
42 function App () {
43   const [estado, dispatch] = useReducer(reducer, estadoInicial)
44
45   useEffect(() => {···
55   }, [])
56
57   useEffect(() => {···
64   }, [estado.token])
```

**Fonte**: O próprio autor, 2020

**Figura 66** - Código refatorado de App de Lembretes II

```
68     <div className='container is-fluid'>
69       <Message header="UFSC - CTC - INE - INE5646 :: App lembretes">
70           <Login onToken={token => dispatch({type: 'RECEBA_NOVO_TOKEN', token})}
71             onSaiu={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}
72             token={estado.token}
73             tokenDecodificado={estado.tokenDecodificado}/>
74           {
75             estado.token &&
76               <PublicaLembrete token={estado.token}
77                 onTokenInvalido={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}/>
78           }
79           {
80             estado.token &&
81               <MostraLembretes token={estado.token}
82                 onTokenInvalido={() => dispatch({type: 'REGISTRE_USUARIO_SAIU'})}/>
83           }
84       </Message>
85     </div>
86   )
87 }
88
89 export default App
```

**Fonte**: O próprio autor, 2020

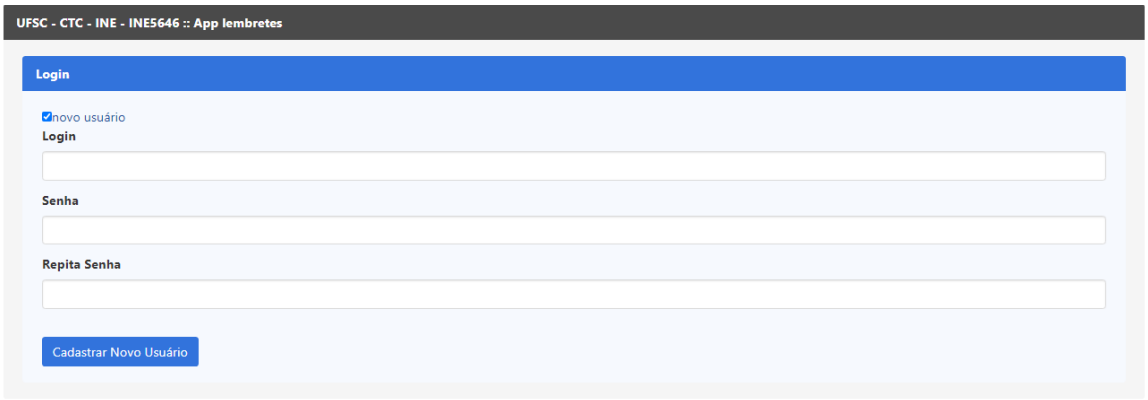
Na **Figura 66**, é possível observar que existem 17 linhas referentes ao código HTML (linha 68 até a 85), 7 tags, além do código pós refactoring possuir um total de 89 linhas.

O componente App apresentou uma redução de 36,36% na quantidade de tags em relação a sua implementação original, redução de 22,73% na quantidade de linhas HTML e redução de 4.30% da quantidade total de linhas do componente. Observa-se que foi utilizado apenas o componente Message no *refactoring* de App.

### *5.1.1.2 Login*

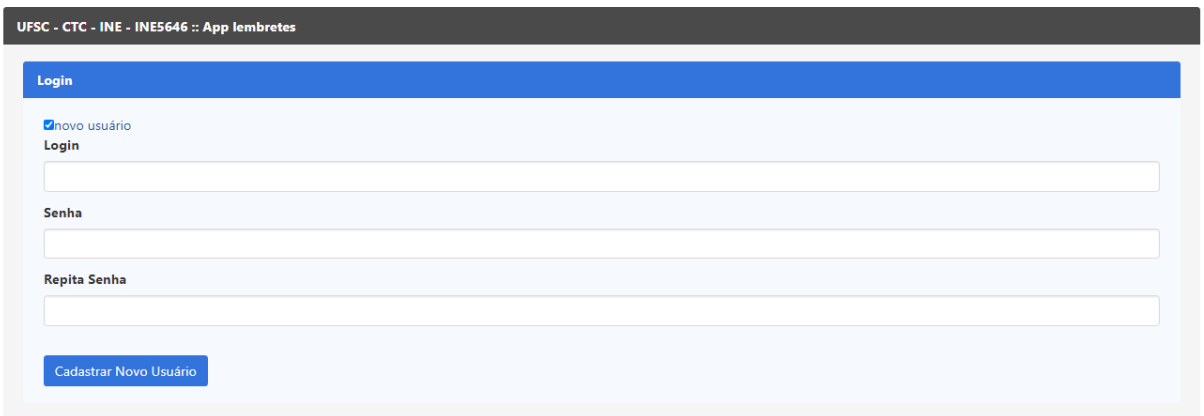
No *refactoring* de Login, utilizou-se de cinco componentes oferecidos pela presente solução, que são: Message, CheckBox, Field, Notification e Button.

**Figura 67** - Imagem original componente Login




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

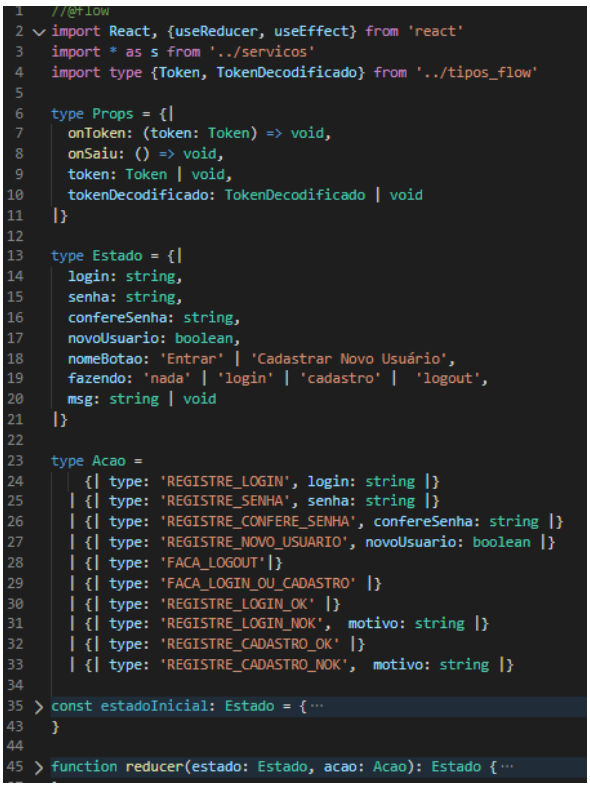
**Figura 68** - Imagem pós refactoring componente Login




**Fonte**: O próprio autor, 2020

Na **Figura *67*** e **Figura 68** é possível observar que o resultado gráfico é o mesmo após o refactoring.

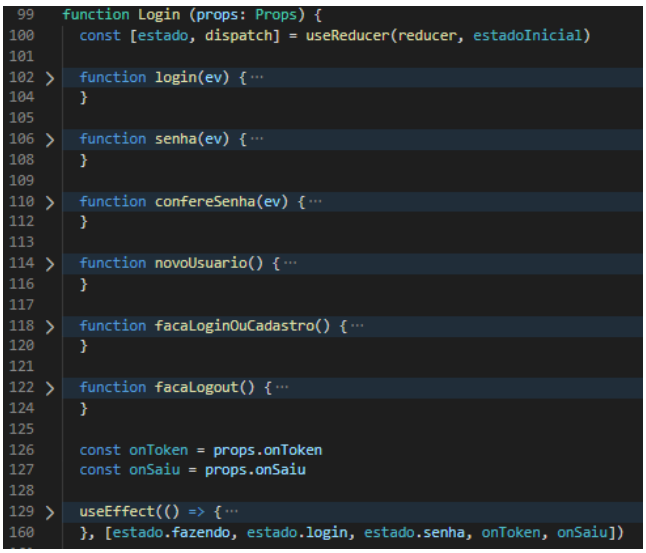
**Figura 69** - Código componente Login de Lembretes I



```
//@flow
import React, {useReducer, useEffect} from 'react'
import * as s from '../servicos'
import type {Token, TokenDecodificado} from '../tipos_flow'

type Props = {|
  onToken: (token: Token) => void,
  onSaiu: () => void,
  token: Token | void,
  tokenDecodificado: TokenDecodificado | void
|}

type Estado = {|
  login: string,
  senha: string,
  confereSenha: string,
  novoUsuario: boolean,
  nomeBotao: 'Entrar' | 'Cadastrar Novo Usuário',
  fazendo: 'nada' | 'login' | 'cadastro' |  'logout',
  msg: string | void
|}

type Acao =
    {| type: 'REGISTRE_LOGIN', login: string |}
  | {| type: 'REGISTRE_SENHA', senha: string |}
  | {| type: 'REGISTRE_CONFERE_SENHA', confereSenha: string |}
  | {| type: 'REGISTRE_NOVO_USUARIO', novoUsuario: boolean |}
  | {| type: 'FACA_LOGOUT'|}
  | {| type: 'FACA_LOGIN_OU_CADASTRO' |}
  | {| type: 'REGISTRE_LOGIN_OK' |}
  | {| type: 'REGISTRE_LOGIN_NOK',  motivo: string |}
  | {| type: 'REGISTRE_CADASTRO_OK' |}
  | {| type: 'REGISTRE_CADASTRO_NOK',  motivo: string |}

const estadoInicial: Estado = {···
}

function reducer(estado: Estado, acao: Acao): Estado {···
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 70** - Código componente Login de Lembretes II



```
function Login (props: Props) {
  const [estado, dispatch] = useReducer(reducer, estadoInicial)

  function login(ev) {···
  }

  function senha(ev) {···
  }

  function confereSenha(ev) {···
  }

  function novoUsuario() {···
  }

  function facaLoginOuCadastro() {···
  }

  function facaLogout() {···
  }

  const onToken = props.onToken
  const onSaiu = props.onSaiu

  useEffect(() => {···
  }, [estado.fazendo, estado.login, estado.senha, onToken, onSaiu])
```

**Fonte**:https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 71** - Código componente Login de Lembretes III

```
let conteudo
if (props.tokenDecodificado === undefined) {
  conteudo =
      <div className='message is-link'>
        <div className='message-header'>Login</div>
        <div className='message-body'>

          <label className='checkbox'>
            <input type='checkbox'
              value={estado.novoUsuario}
              onChange={novoUsuario}/>novo usuário
          </label>
          <div className='field'>
            <label className='label'>Login</label>
            <div className='control'>
              <input className='input' type='text'
                value={estado.login} onChange={login}/>
            </div>
          </div>
          <div className='field'>
            <label className='label'>Senha</label>
            <div className='control'>
              <input className='input' type='password'
                value={estado.senha}
                onChange={senha}/>
            </div>
          </div>
          {
            estado.novoUsuario &&
          <div className='field'>
            <label className='label'>Repita Senha</label>
            <div className='control'>
              <input className='input' type='password'
                value={estado.confereSenha}
                onChange={confereSenha}/>
            </div>
          </div>
          }
          <br/>
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 72** - Código componente Login de Lembretes IV

```
204                <button className='button is-link' onClick={facaLoginOuCadastro}>
205                  {estado.nomeBotao}
206                </button>
207                {
208                  estado.msg &&
209                <div className='notification is-warning'>{estado.msg}</div>
210                }
211              </div>
212            </div>
213      } else {
214        conteudo =
215            <div className='message is-info'>
216              <div className='message-header'>
217                Usuário logado: {props.tokenDecodificado.login}
218              </div>
219              <div className='message-body'>
220                <button className='button is-link' onClick={facaLogout}>Sair</button>
221              </div>
222            </div>
223      }
224
225      return (
226        <div>
227          {conteudo}
228        </div>
229      )
230    }
231
232
233    export default Login
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

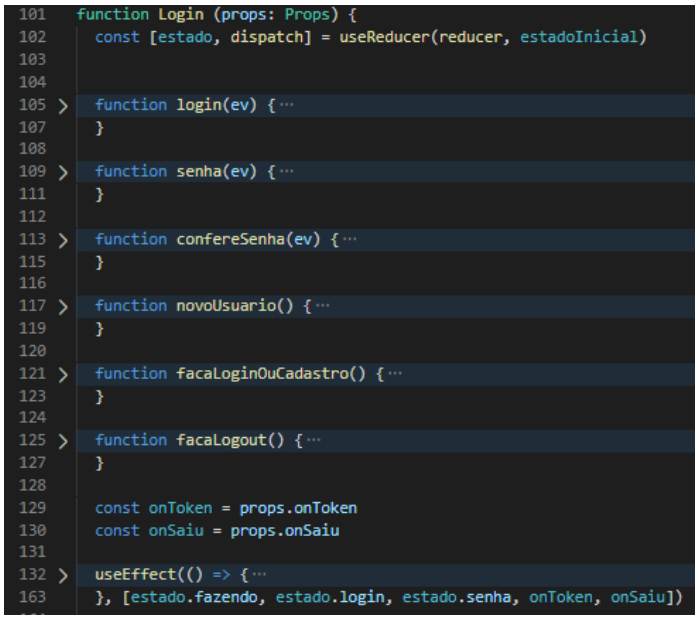
Da **Figura *69*** até a figura ***Figura 72*** é possível observar que existem 56 linhas referente ao código HTML (168-212,215-222 e 226-228), 45 tags, além de possuir um total de 233 linhas no código original completo.

**Figura 73** - Código refatorado componente Login Lembretes I

```
//@flow
import React, {useReducer, useEffect} from 'react'
import * as s from '../servicos'
import {Message,CheckBox,Field,Notification,Button} from 'assemble-react-bulma';
import type {Token, TokenDecodificado} from '../tipos_flow'

type Props = {|
  onToken: (token: Token) => void,
  onSaiu: () => void,
  token: Token | void,
  tokenDecodificado: TokenDecodificado | void
|}

type Estado = {|
  login: string,
  senha: string,
  confereSenha: string,
  novoUsuario: boolean,
  nomeBotao: 'Entrar' | 'Cadastrar Novo Usuário',
  fazendo: 'nada' | 'login' | 'cadastro' |  'logout',
  msg: string | void
|}

type Acao =
    {| type: 'REGISTRE_LOGIN', login: string |}
  | {| type: 'REGISTRE_SENHA', senha: string |}
  | {| type: 'REGISTRE_CONFERE_SENHA', confereSenha: string |}
  | {| type: 'REGISTRE_NOVO_USUARIO', novoUsuario: boolean |}
  | {| type: 'FACA_LOGOUT'|}
  | {| type: 'FACA_LOGIN_OU_CADASTRO' |}
  | {| type: 'REGISTRE_LOGIN_OK' |}
  | {| type: 'REGISTRE_LOGIN_NOK',  motivo: string |}
  | {| type: 'REGISTRE_CADASTRO_OK' |}
  | {| type: 'REGISTRE_CADASTRO_NOK',  motivo: string |}

const estadoInicial: Estado = {···
}

function reducer(estado: Estado, acao: Acao): Estado {···
}
```

**Fonte**: O próprio autor, 2020

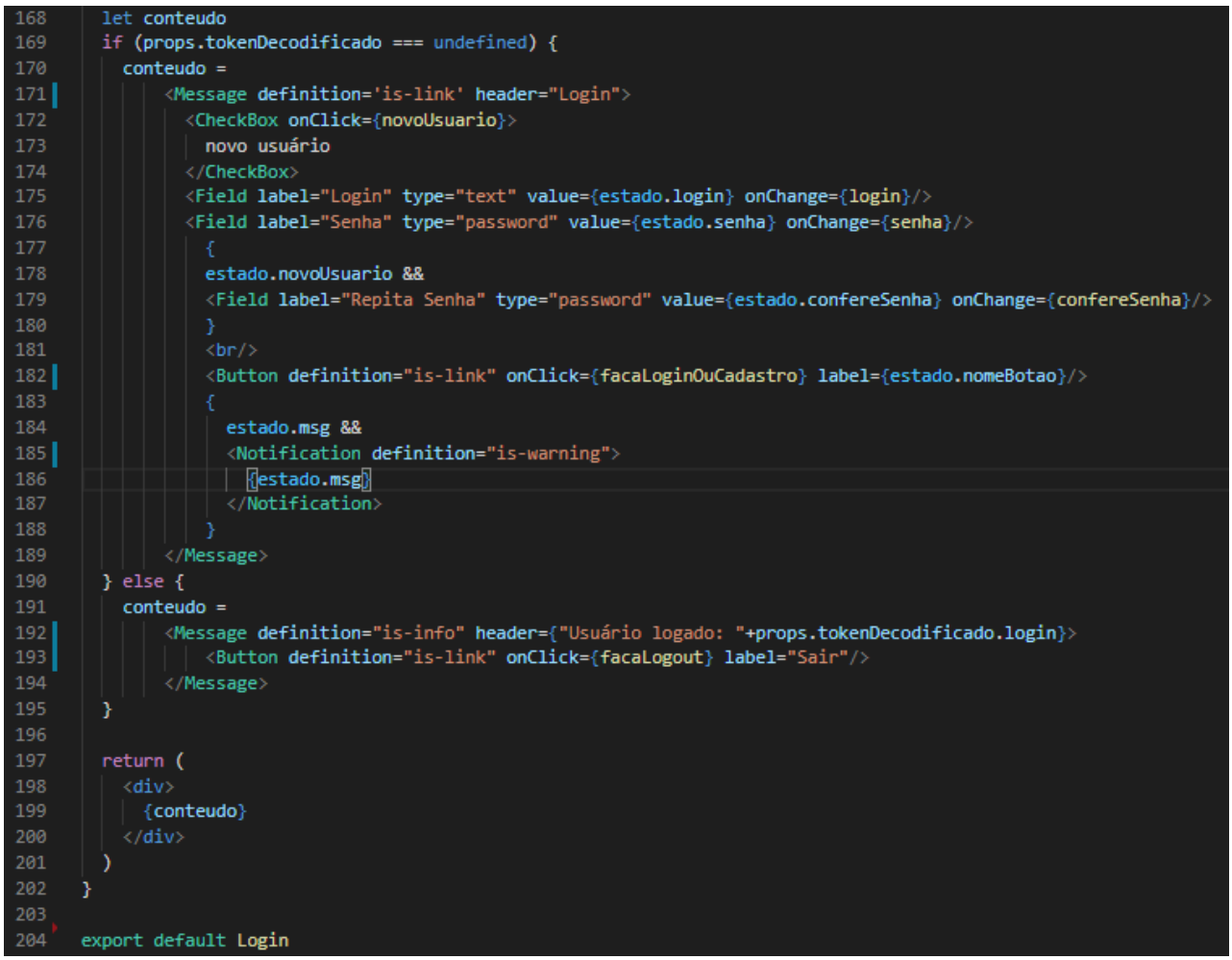
**Figura 74** - Código refatorado componente Login Lembretes II



```
function Login (props: Props) {
  const [estado, dispatch] = useReducer(reducer, estadoInicial)


  function login(ev) {···
  }

  function senha(ev) {···
  }

  function confereSenha(ev) {···
  }

  function novoUsuario() {···
  }

  function facaLoginOuCadastro() {···
  }

  function facaLogout() {···
  }

  const onToken = props.onToken
  const onSaiu = props.onSaiu

  useEffect(() => {···
  }, [estado.fazendo, estado.login, estado.senha, onToken, onSaiu])
```

**Fonte**: O próprio autor, 2020

**Figura 75** - Código refatorado componente Login Lembretes III



```
168   let conteudo
169   if (props.tokenDecodificado === undefined) {
170     conteudo =
171         <Message definition='is-link' header="Login">
172           <CheckBox onClick={novoUsuario}>
173             novo usuário
174           </CheckBox>
175           <Field label="Login" type="text" value={estado.login} onChange={login}/>
176           <Field label="Senha" type="password" value={estado.senha} onChange={senha}/>
177             {
178             estado.novoUsuario &&
179             <Field label="Repita Senha" type="password" value={estado.confereSenha} onChange={confereSenha}/>
180             }
181             <br/>
182             <Button definition="is-link" onClick={facaLoginOuCadastro} label={estado.nomeBotao}/>
183             {
184               estado.msg &&
185               <Notification definition="is-warning">
186                 {estado.msg}
187               </Notification>
188             }
189         </Message>
190   } else {
191     conteudo =
192         <Message definition="is-info" header={"Usuário logado: "+props.tokenDecodificado.login}>
193           <Button definition="is-link" onClick={facaLogout} label="Sair"/>
194         </Message>
195   }
196
197   return (
198     <div>
199       {conteudo}
200     </div>
201   )
202 }
203
204 export default Login
```

**Fonte**: O próprio autor, 2020

Na Figura 73,Figura 74 e Figura 75 é possível observar que existem 25 linhas referente ao código HTML (171-189, 192-194 e 198-200), além de possui um total de 206 linhas do código refatorado.

O componente Message simplificou 3 tags em uma, é possível observar que as tags div contendo os classnames “message is-link”,”message-header” e “message-body” na ***Figura 71*** e ***Figura 72*** já não são encontrados após o *refactoring* (**Figura *75***), pois o componente Message englobou todas as tags antes mencionadas na sua implementação. Também é possível fazer uma relação de equivalência entre o código original e o resultado pós *refactoring*, ao observar que ao invés de informar “message is-info” ou “message is-link” bastou chamar o próprio componente e informar definition(*props*),“message-header” bastou informar header(*props*) e ao invés de

informar “message-body” precisou-se apenas passar um *children* para o componente Message.

O componente Checkbox diminuiu de duas tags para uma e retirou uma certa verbosidade ao criar um checkbox. Caso seja utilizada a forma descrita na **Figura *71*** é necessário informar a tag label com um className=”label”, informar uma tag input do tipo checkbox, fornecer o nome do checkbox através de value e informar o evento, já ao utilizar o componente CheckBox mostrado na **Figura *75*** bastou invocar CheckBox através de uma única tag, passar um método onClick (que nessa situação equivale a onChange) e passar o valor do seu nome como *children* ao invés de utilizar o atributo value. Na **Figura *71*** que a maior parte do código HTML é oriundo do uso do elemento “field” e seus dependentes “input”, “label” e “control”. Na presente solução existe um componente chamado Field, criado especialmente para essa situação. Essa redução é consequência de uma maior abstração de elementos que existe no componente, a tag “label” é substituída pelo atributo label (*props*) do componente, a tag “control” é implementada internamente então não é necessário informa-la (apesar de ser customizável), o type é informado pois Field suporta essa customização e value também é informado.

O componente Login apresentou uma redução de 64.44% na quantidade de tags, redução de 55.36% na quantidade de linhas HTML e redução de 11,59% do total de linhas do componente. No caso de Login, utilizou-se 5 componentes da presente solução.

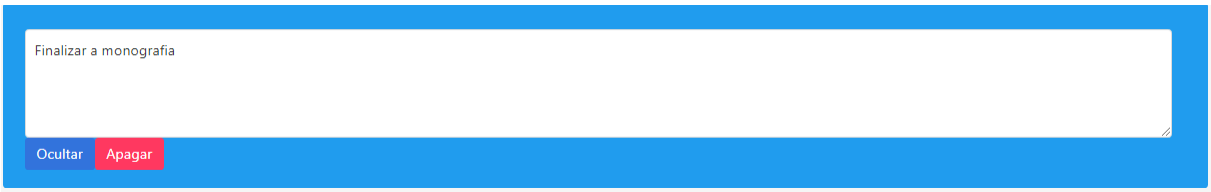
#### *5.1.1.3 MostraLembrete*

No *refactoring* de MostraLembretes utilizou-se os componentes Notification, TextArea e Button.

**Figura 76** - Imagem original componente MostraLembrete


**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 77** - Imagem após refactoring do componente MostraLembrete




**Fonte**: O próprio autor, 2020

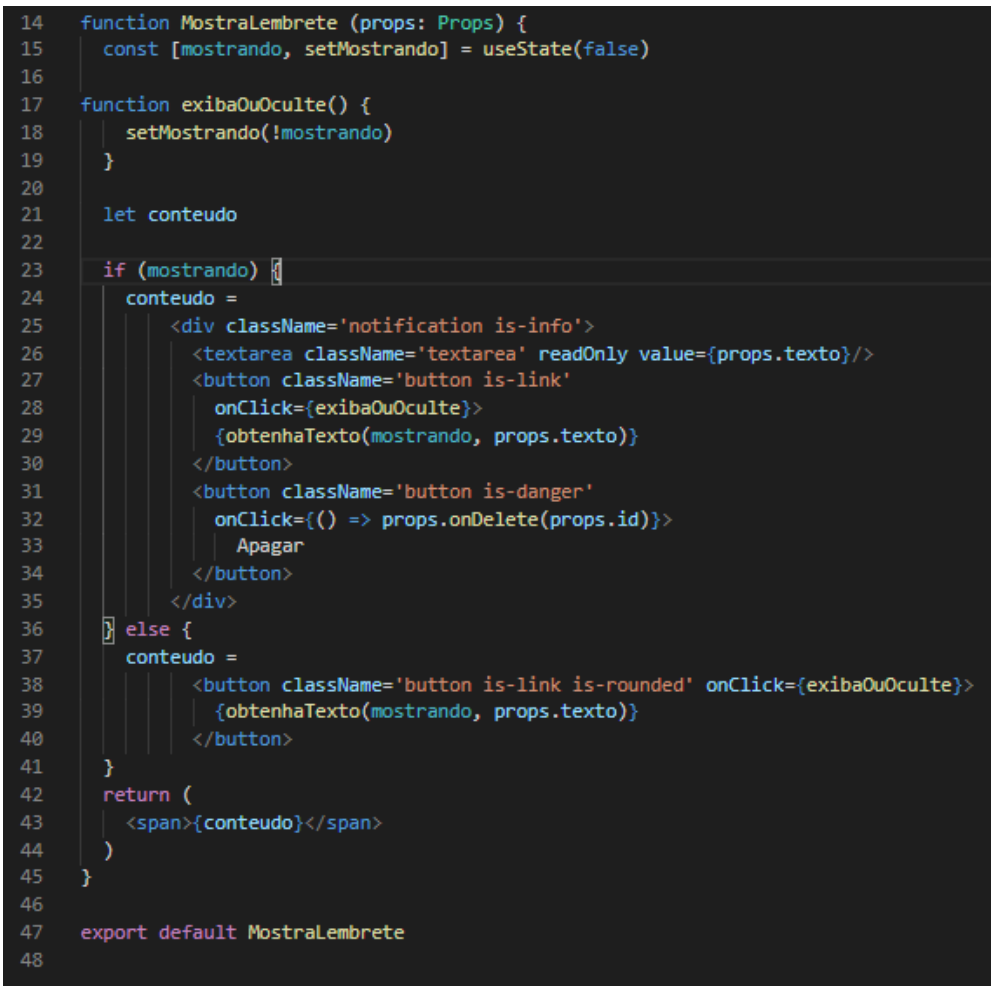
Observa-se que os resultado gráfico do refactoring é o mesmo da interface original, conforme apresentado na Figura 76 e Figura 77.

**Figura 78** - Código original componente MostraLembrete I

```
1  //@flow
2  import React, {useState} from 'react'
3
4  type Props = {|
5    id: string,
6    texto: string,
7    onDelete: string => void
8  |}
9
10 function obtenhaTexto(mostrando: boolean, texto: string): string {
11   return mostrando ?  'Ocultar' : `${texto.substring(0,10)}...`
12 }
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 79** - Código original componente MostraLembrete II

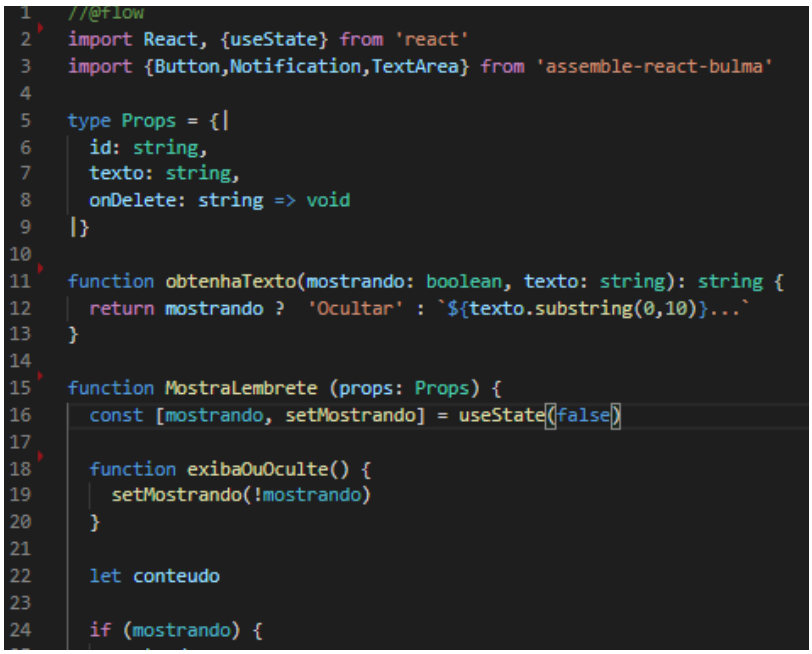


```
14  function MostraLembrete (props: Props) {
15    const [mostrando, setMostrando] = useState(false)
16
17  function exibaOuOculte() {
18      setMostrando(!mostrando)
19    }
20
21    let conteudo
22
23    if (mostrando) {
24      conteudo =
25          <div className='notification is-info'>
26            <textarea className='textarea' readOnly value={props.texto}/>
27            <button className='button is-link'
28              onClick={exibaOuOculte}>
29              {obtenhaTexto(mostrando, props.texto)}
30            </button>
31            <button className='button is-danger'
32              onClick={() => props.onDelete(props.id)}>
33                Apagar
34            </button>
35          </div>
36    } else {
37      conteudo =
38            <button className='button is-link is-rounded' onClick={exibaOuOculte}>
39              {obtenhaTexto(mostrando, props.texto)}
40            </button>
41    }
42    return (
43      <span>{conteudo}</span>
44    )
45  }
46
47  export default MostraLembrete
48
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

Na **Figura *78*** e **Figura *79***, é possível observar que existem 14 linhas HTML(25-35,38-40 e 43) , 11 tags e uma quantidade total de 47 linhas do código original do componente.

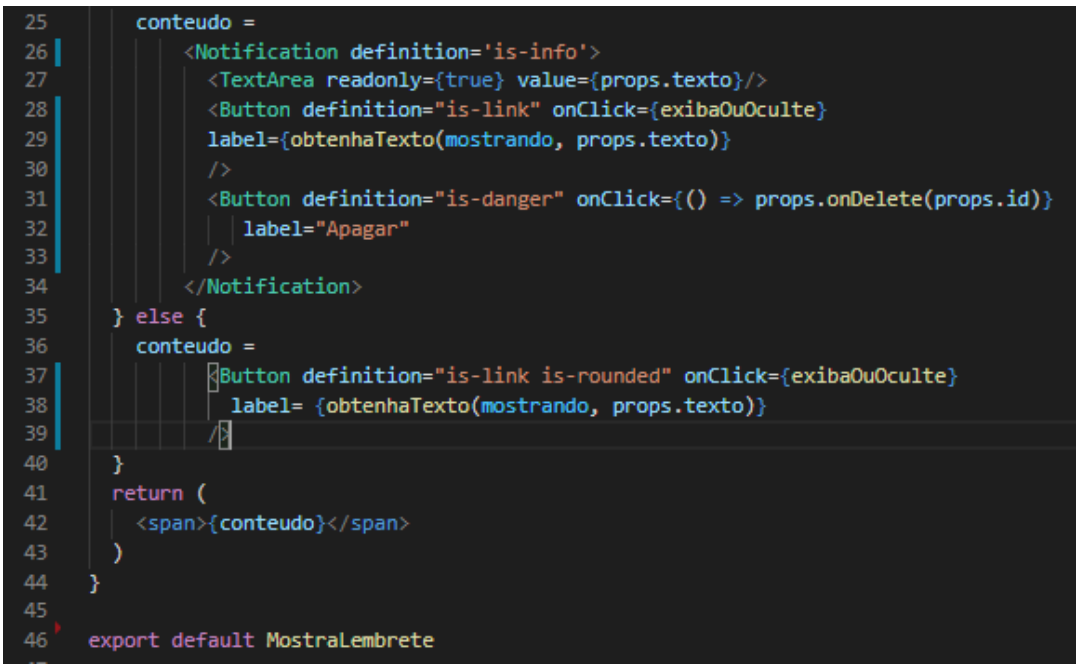
**Figura 80** - Código refatorado componente MostraLembrete I



```
//@flow
import React, {useState} from 'react'
import {Button,Notification,TextArea} from 'assemble-react-bulma'

type Props = {|
  id: string,
  texto: string,
  onDelete: string => void
|}

function obtenhaTexto(mostrando: boolean, texto: string): string {
  return mostrando ?  'Ocultar' : `${texto.substring(0,10)}...`
}

function MostraLembrete (props: Props) {
  const [mostrando, setMostrando] = useState(false)

  function exibaOuOculte() {
    setMostrando(!mostrando)
  }

  let conteudo

  if (mostrando) {
```

**Fonte**: O próprio autor, 2020

**Figura 81** - Código refatorado componente MostraLembrete II



```
    conteudo =
        <Notification definition='is-info'>
          <TextArea readonly={true} value={props.texto}/>
          <Button definition="is-link" onClick={exibaOuOculte}
          label={obtenhaTexto(mostrando, props.texto)}
          />
          <Button definition="is-danger" onClick={() => props.onDelete(props.id)}
            label="Apagar"
          />
        </Notification>
  } else {
    conteudo =
          <Button definition="is-link is-rounded" onClick={exibaOuOculte}
            label= {obtenhaTexto(mostrando, props.texto)}
          />
  }
  return (
    <span>{conteudo}</span>
  )
}

export default MostraLembrete
```

**Fonte**: O próprio autor, 2020

Na **Figura *80*** e **Figura *81*** é possível observar que existem 13 linhas HTML(26-34,37-39 e 42), 8 tags e uma quantidade total de 47 linhas do código refatorado do componente.

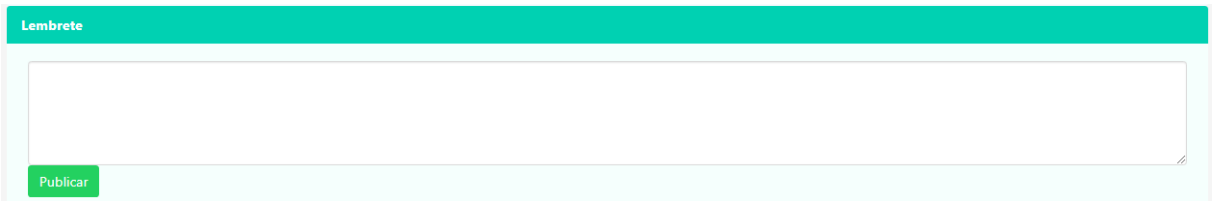
A redução de código nesse componente foi irrisória, a quantidade de tags são as mesmas e a quantidade de total de linha diminuiu uma unidade no código refatorado. A motivação da não redução de tags é porque os componentes utilizados podem ser descritos com apenas uma tag, componentes simples que não necessitam de uma grande quantidade de código, como é o caso de todos os componentes envolvidos no *refactoring* de MostraLembretes. Uma forma alternativa de entender a não redução das tags nesse exemplo é pensar que os componentes envolvidos não apresentam uma grande abstração, pois não necessitam de grande quantidade de código.

O componente MostraLembretes apresentou uma redução de 27.27% no número de tags, 0% no número de linhas totais e 7,14% no número de linhas HTML. Observa-se também que a redução de 0% no número de linhas totais é ocasionada pelo componente MostraLembretes utilizar do modo mais básico possível os componentes da presente solução, o que gerou uma equivalência na quantidade de tags

#### *5.1.1.4 PublicaLembretes*

No *refactoring* de PublicaLembretes utilizou-se os componentes Button, TextArea e Message.

**Figura 82** - Imagem original do componente PublicaLembrete




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 83** - Imagem após refactoring do componente PublicaLembrete


**Fonte**: O próprio autor, 2020

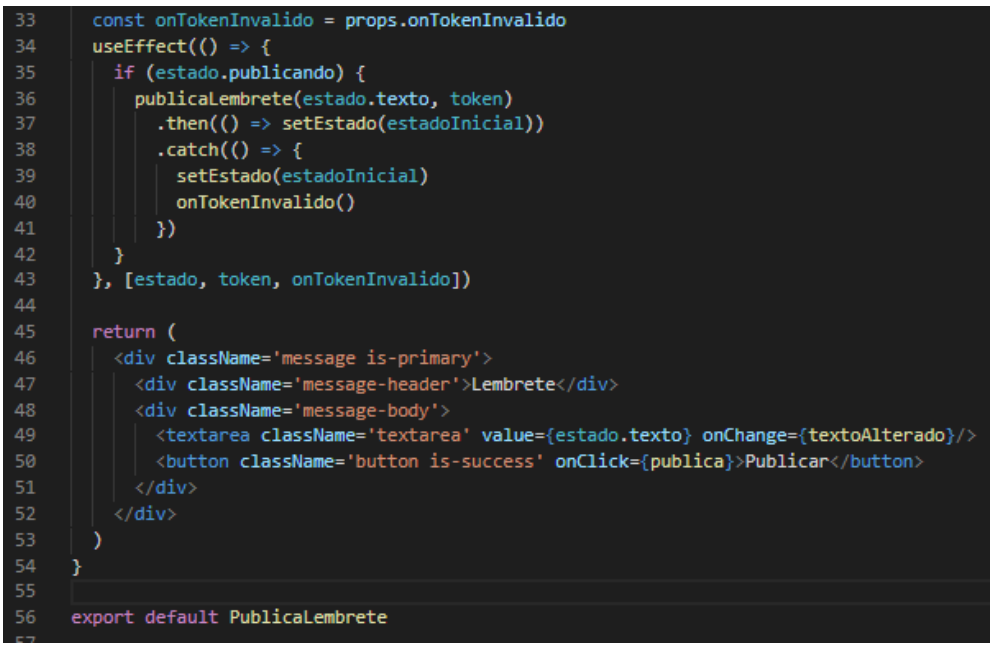
Na **Figura *82*** e **Figura *83*** é possível observar que não existem mudanças nos resultados gráficos do componente original e do refatorado.

**Figura 84** - Código original do componente PublicaLembrete I

```
//@flow
import React, {useState, useEffect} from 'react'
import {publicaLembrete} from '../servicos'
import type {Token} from '../tipos_flow'

type Props = {|
  token: Token,
  onTokenInvalido: void => void
|}

type Estado = {|
  texto: string,
  publicando: boolean
|}

const estadoInicial: Estado = {
  texto: '',
  publicando: false
}

function PublicaLembrete (props: Props) {
  const [estado: Estado, setEstado: Estado => void] = useState(estadoInicial)

  function textoAlterado(ev) {
    setEstado({texto: ev.target.value, publicando: false})
  }

  function publica() {
    setEstado({texto: estado.texto, publicando: true})
  }

  const token = props.token
  const onTokenInvalido = props.onTokenInvalido
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

**Figura 85** - Código original do componente PublicaLembrete II



```
33    const onTokenInvalido = props.onTokenInvalido
34    useEffect(() => {
35      if (estado.publicando) {
36        publicaLembrete(estado.texto, token)
37          .then(() => setEstado(estadoInicial))
38          .catch(() => {
39            setEstado(estadoInicial)
40            onTokenInvalido()
41          })
42      }
43    }, [estado, token, onTokenInvalido])
44
45    return (
46      <div className='message is-primary'>
47        <div className='message-header'>Lembrete</div>
48        <div className='message-body'>
49          <textarea className='textarea' value={estado.texto} onChange={textoAlterado}/>
50          <button className='button is-success' onClick={publica}>Publicar</button>
51        </div>
52      </div>
53    )
54  }
55
56  export default PublicaLembrete
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/05-app_lembretes

Na **Figura *85*** é possível observar que existem 7 linhas HTML (46-52), 9 tags, além de uma quantidade total de 56 linhas do código original do componente.

**Figura 86** - Código refatorado componente PublicaLembrete I

```
1   //@flow
2   import React, {useState, useEffect} from 'react'
3   import {publicaLembrete} from '../servicos'
4   import {Message,TextArea,Button} from 'assemble-react-bulma';
5   import type {Token} from '../tipos_flow'
6
7   type Props = {|
8     token: Token,
9     onTokenInvalido: void => void
10  |}
11
12  type Estado = {|
13    texto: string,
14    publicando: boolean
15  |}
16
17  const estadoInicial: Estado = {
18    texto: '',
19    publicando: false
20  }
21
22  function PublicaLembrete (props: Props) {
23    const [estado: Estado, setEstado: Estado => void] = useState(estadoInicial)
24
25    function textoAlterado(ev) {
26      setEstado({texto: ev.target.value, publicando: false})
27    }
28
29    function publica() {
30      setEstado({texto: estado.texto, publicando: true})
31    }
32
33    const token = props.token
34    const onTokenInvalido = props.onTokenInvalido
```

**Fonte**: O próprio autor, 2020

**Figura 87** - Código refatorado do componente PublicaLembrete II

```
35  useEffect(() => {
36    if (estado.publicando) {
37      publicaLembrete(estado.texto, token)
38        .then(() => setEstado(estadoInicial))
39        .catch(() => {
40          setEstado(estadoInicial)
41          onTokenInvalido()
42        })
43    }
44  }, [estado, token, onTokenInvalido])
45
46  return (
47    <Message definition='is-primary' header="Lembrete">
48        <TextArea value={estado.texto} onChange={textoAlterado}/>
49        <Button definition='is-success' onClick={publica} label="Publicar"/>
50    </Message>
51  )
52 }
53
54
55 export default PublicaLembrete
```

**Fonte**: O próprio autor, 2020

Na Figura 87, é possível observar que existem 4 linhas HTML (47-50), 5 tags, além de uma quantidade total de 55 linhas do código refatorado do componente.

O componente Message (igual detalhado no *refactoring* de Login), englobou três tags em apenas uma, substituiu o valor de “message-header” pela propriedade header (props) e tornou *children* equivalente a “message-body”, excluindo a necessidade de uma tag para “message-body”.

O componente PublicaLembretes apresentou uma redução de 55.56% no número de tags, 42,86% no número de linhas HTML e 1,79% no total de linhas do componente.

### *5.1.1.5 Métricas refactoring*

No presente subtópico são apresentadas as métricas do refactoring aplicados nos componentes de App Lembretes.

**Tabela 8** - Legenda App Lembretes

| Sigla | Descrição |
|---|---|
| L O | Total de linhas do código original |
| L R | Total de linhas do código refatorado |
| TO | Total de tags do código original |
| TR | Total de tags do código refatorado |
| LHO | Total de linhas HTML do código original |
| LHR | Total de linhas HTML do código refatorado |
| %LH | Porcentagem de redução do código HTML |
| %LT | Porcentagem de redução do código total |
| %T | Porcentagem de redução do número de tags |

**Fonte**: O próprio autor, 2020

**Tabela 9** - Métrica refactoring App Lembretes

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| App | 93 | 89 | 11 | 7 | 22 | 17 | 4.30 | 36.36 | 22.73 |
| Login | 233 | 206 | 45 | 16 | 56 | 25 | 11.59 | 64.44 | 55.36 |
| MostraLembretes | 47 | 47 | 11 | 8 | 14 | 13 | 0.00 | 27.27 | 7.14 |
| PublicaLembrete | 56 | 55 | 9 | 4 | 7 | 4 | 1.79 | 55.56 | 42.86 |

**Fonte**: O próprio autor, 2020

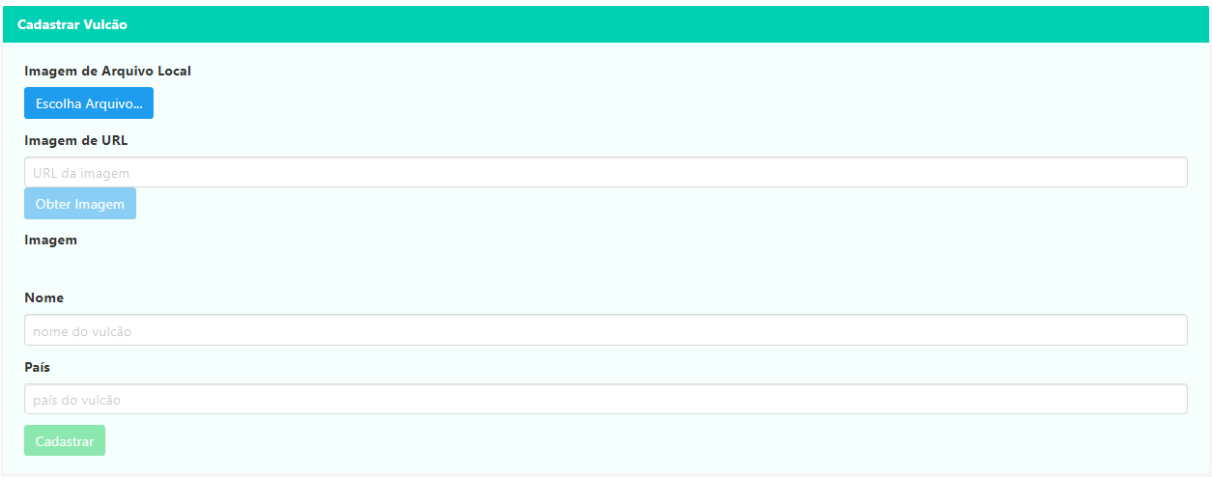
### 5.1.2 App Vulcões

Essa aplicação trata de um simples sistema para cadastrar vulcões pelo nome e país, além de adicionar uma foto para cada vulcão. É possível cadastrar usuário e efetuar o login, além de poder cadastrar, excluir e exibir os vulcões. Os componentes React utilizados nessa aplicação são App, CadastraVulcao, ConexaoComBanco, Login, MostraVulcao e MsgModal.

#### *5.1.2.1 CadastraVulcao Original*

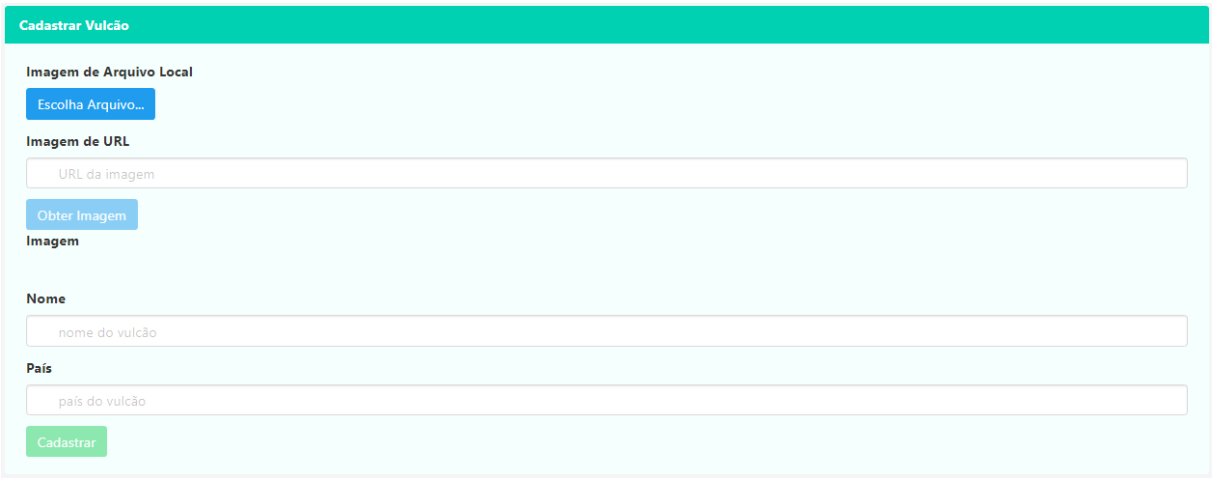
No *refactoring* de CadastraVulcao utilizou-se os componentes Message, FileButton, Field, FieldContent e Button.

**Figura 88** - Imagem original componente CadastraVulcao




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/09-app_vulcoes

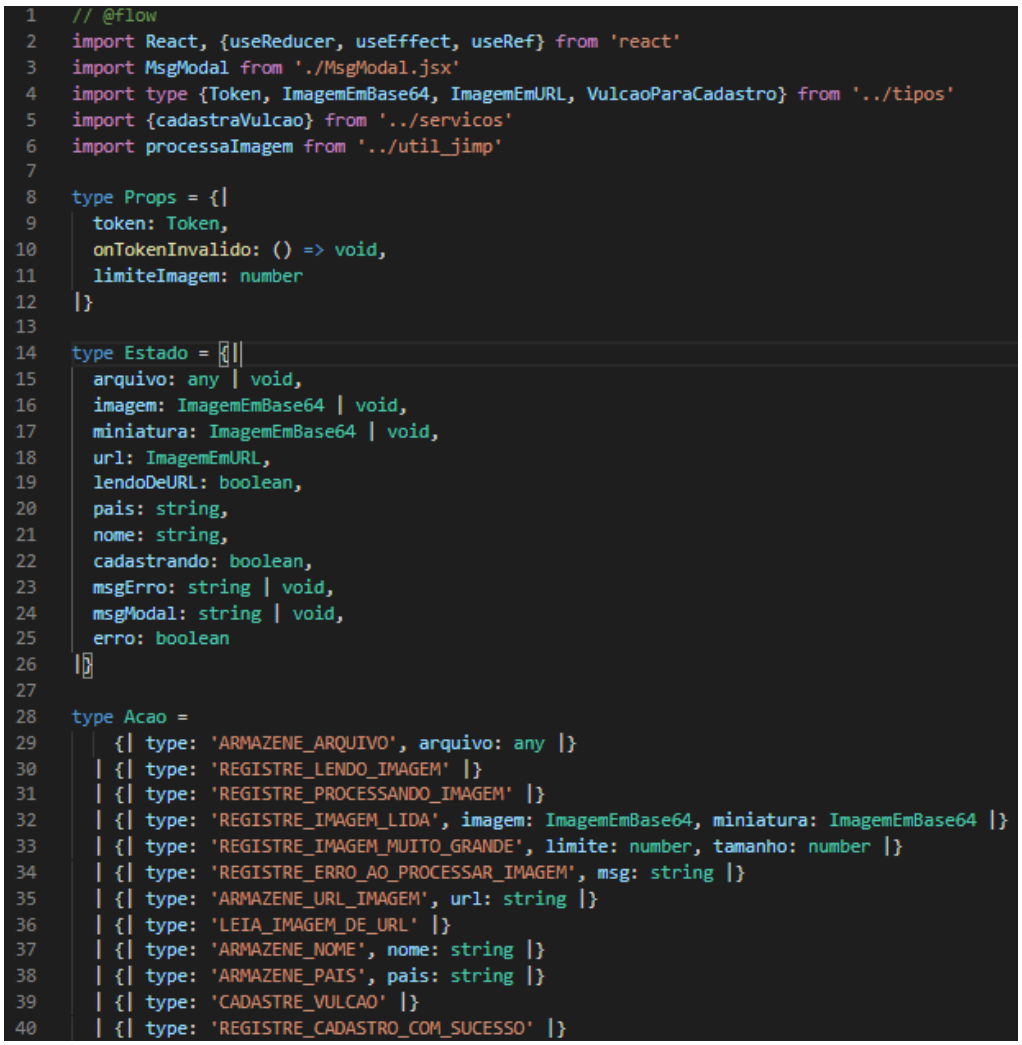
**Figura 89** - Imagem componente CadastraVulcao refatorado




**Fonte**: O próprio autor, 2020

Observa-se que o resultado após o refactoring é igual ao resultado gráfico original do componente CadastraVulcao, conforme apresentado na **Figura *88*** e **Figura *89***.

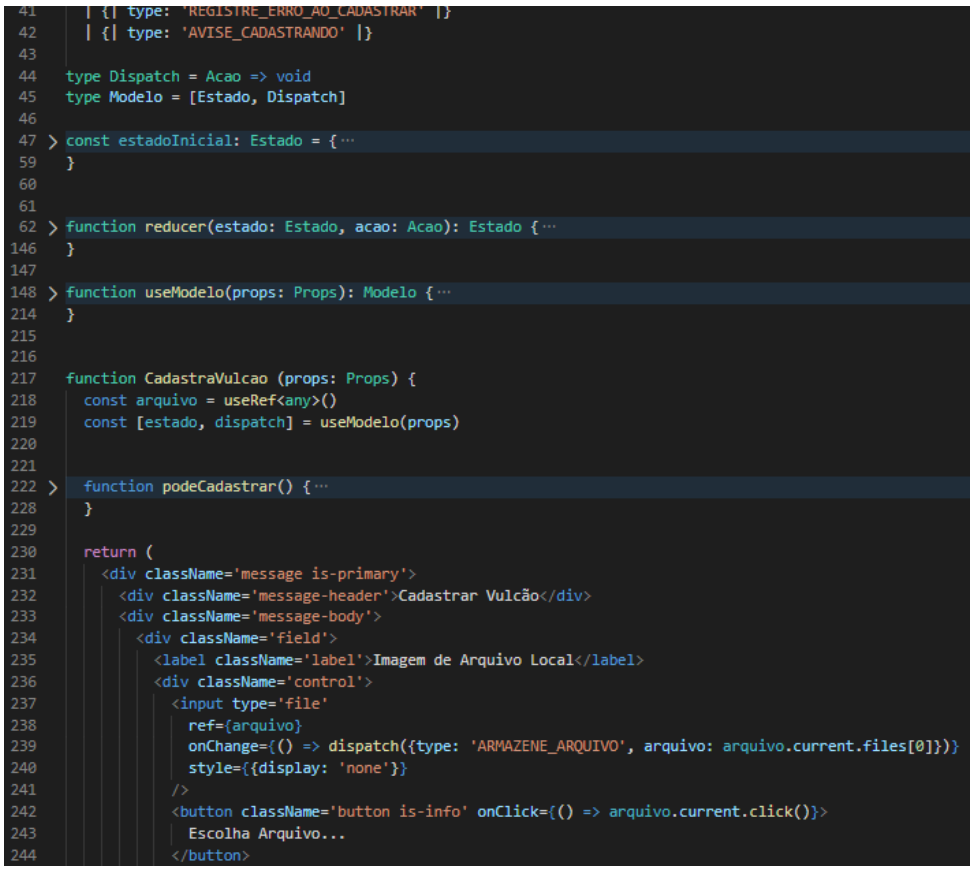
**Figura 90** - Código original componente CadastraVulcao I



```
// @flow
import React, {useReducer, useEffect, useRef} from 'react'
import MsgModal from './MsgModal.jsx'
import type {Token, ImagemEmBase64, ImagemEmURL, VulcaoParaCadastro} from '../tipos'
import {cadastraVulcao} from '../servicos'
import processaImagem from '../util_jimp'

type Props = {|
  token: Token,
  onTokenInvalido: () => void,
  limiteImagem: number
|}

type Estado = {|
  arquivo: any | void,
  imagem: ImagemEmBase64 | void,
  miniatura: ImagemEmBase64 | void,
  url: ImagemEmURL,
  lendoDeURL: boolean,
  pais: string,
  nome: string,
  cadastrando: boolean,
  msgErro: string | void,
  msgModal: string | void,
  erro: boolean
|}

type Acao =
    {| type: 'ARMAZENE_ARQUIVO', arquivo: any |}
  | {| type: 'REGISTRE_LENDO_IMAGEM' |}
  | {| type: 'REGISTRE_PROCESSANDO_IMAGEM' |}
  | {| type: 'REGISTRE_IMAGEM_LIDA', imagem: ImagemEmBase64, miniatura: ImagemEmBase64 |}
  | {| type: 'REGISTRE_IMAGEM_MUITO_GRANDE', limite: number, tamanho: number |}
  | {| type: 'REGISTRE_ERRO_AO_PROCESSAR_IMAGEM', msg: string |}
  | {| type: 'ARMAZENE_URL_IMAGEM', url: string |}
  | {| type: 'LEIA_IMAGEM_DE_URL' |}
  | {| type: 'ARMAZENE_NOME', nome: string |}
  | {| type: 'ARMAZENE_PAIS', pais: string |}
  | {| type: 'CADASTRE_VULCAO' |}
  | {| type: 'REGISTRE_CADASTRO_COM_SUCESSO' |}
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/09-app_vulcoes

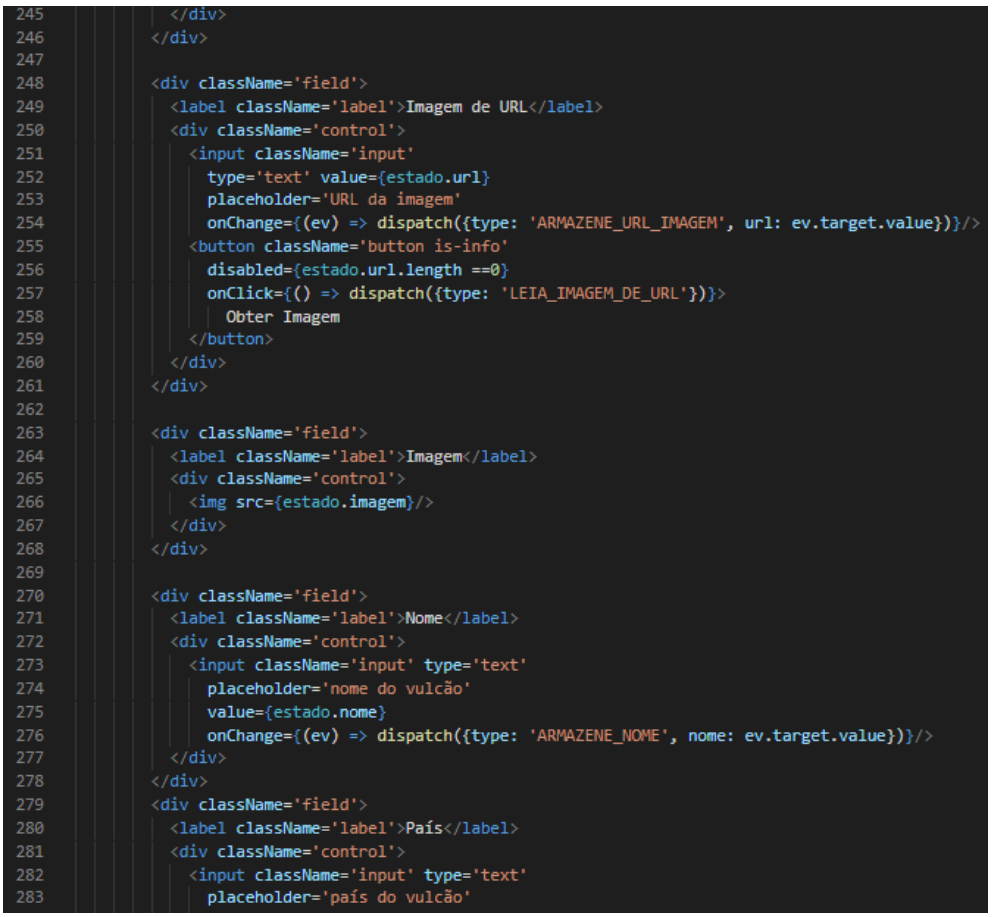
**Figura 91** - Código original componente CadastraVulcao II



```
  | {| type: 'REGISTRE_ERRO_AO_CADASTRAR' |}
  | {| type: 'AVISE_CADASTRANDO' |}

type Dispatch = Acao => void
type Modelo = [Estado, Dispatch]

const estadoInicial: Estado = {···
}


function reducer(estado: Estado, acao: Acao): Estado {···
}

function useModelo(props: Props): Modelo {···
}


function CadastraVulcao (props: Props) {
  const arquivo = useRef<any>()
  const [estado, dispatch] = useModelo(props)


  function podeCadastrar() {···
  }

  return (
    <div className='message is-primary'>
      <div className='message-header'>Cadastrar Vulcão</div>
      <div className='message-body'>
        <div className='field'>
          <label className='label'>Imagem de Arquivo Local</label>
          <div className='control'>
            <input type='file'
              ref={arquivo}
              onChange={() => dispatch({type: 'ARMAZENE_ARQUIVO', arquivo: arquivo.current.files[0]})}
              style={{display: 'none'}}
            />
            <button className='button is-info' onClick={() => arquivo.current.click()}>
              Escolha Arquivo...
            </button>
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/09-app_vulcoes

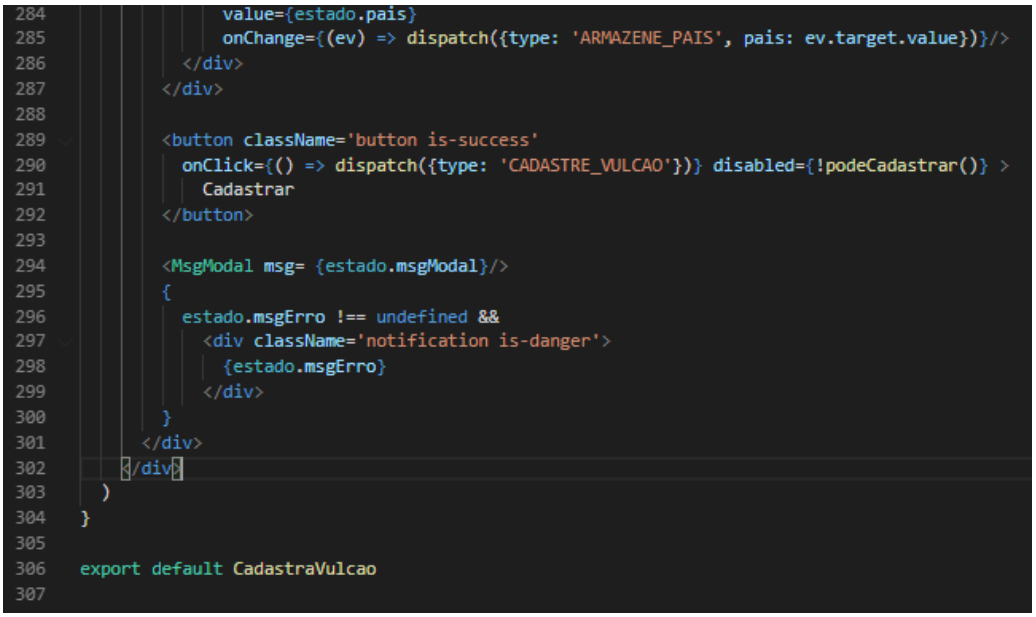
**Figura 92** - Código original componente CadastraVulcao III



```
245                </div>
246              </div>
247
248              <div className='field'>
249                <label className='label'>Imagem de URL</label>
250                <div className='control'>
251                  <input className='input'
252                    type='text' value={estado.url}
253                    placeholder='URL da imagem'
254                    onChange={(ev) => dispatch({type: 'ARMAZENE_URL_IMAGEM', url: ev.target.value})}/>
255                  <button className='button is-info'
256                    disabled={estado.url.length ==0}
257                    onClick={() => dispatch({type: 'LEIA_IMAGEM_DE_URL'})}>
258                      Obter Imagem
259                  </button>
260                </div>
261              </div>
262
263              <div className='field'>
264                <label className='label'>Imagem</label>
265                <div className='control'>
266                  <img src={estado.imagem}/>
267                </div>
268              </div>
269
270              <div className='field'>
271                <label className='label'>Nome</label>
272                <div className='control'>
273                  <input className='input' type='text'
274                    placeholder='nome do vulcão'
275                    value={estado.nome}
276                    onChange={(ev) => dispatch({type: 'ARMAZENE_NOME', nome: ev.target.value})}/>
277                </div>
278              </div>
279              <div className='field'>
280                <label className='label'>País</label>
281                <div className='control'>
282                  <input className='input' type='text'
283                    placeholder='país do vulcão'
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/09-app_vulcoes

**Figura 93** - Código original componente CadastraVulcao IV

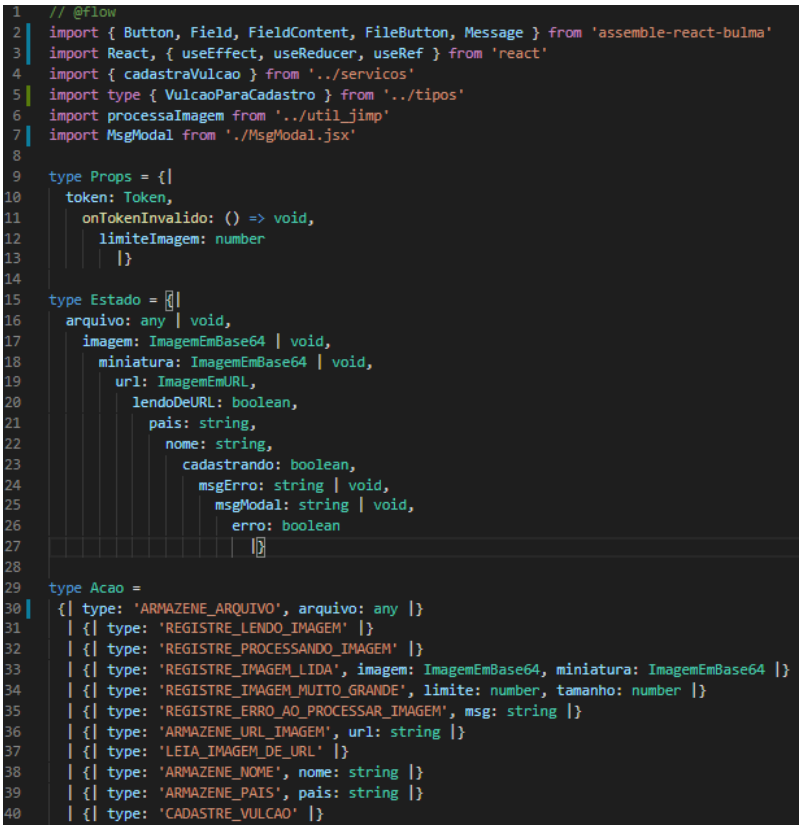


```
284                    value={estado.pais}
285                    onChange={(ev) => dispatch({type: 'ARMAZENE_PAIS', pais: ev.target.value})}/>
286                </div>
287              </div>
288
289              <button className='button is-success'
290                onClick={() => dispatch({type: 'CADASTRE_VULCAO'})} disabled={!podeCadastrar()} >
291                  Cadastrar
292              </button>
293
294              <MsgModal msg= {estado.msgModal}/>
295              {
296                estado.msgErro !== undefined &&
297                  <div className='notification is-danger'>
298                    {estado.msgErro}
299                  </div>
300              }
301            </div>
302          </div>
303        )
304      }
305
306      export default CadastraVulcao
307
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/09-app_vulcoes

Nas figuras **Figura *90*** e **Figura *91*** e **Figura *92*** e **Figura *93*** é possível observar que existem 72 linhas HTML (201-302), 41 tags, além de uma quantidade total de 306 linhas do código original do componente.

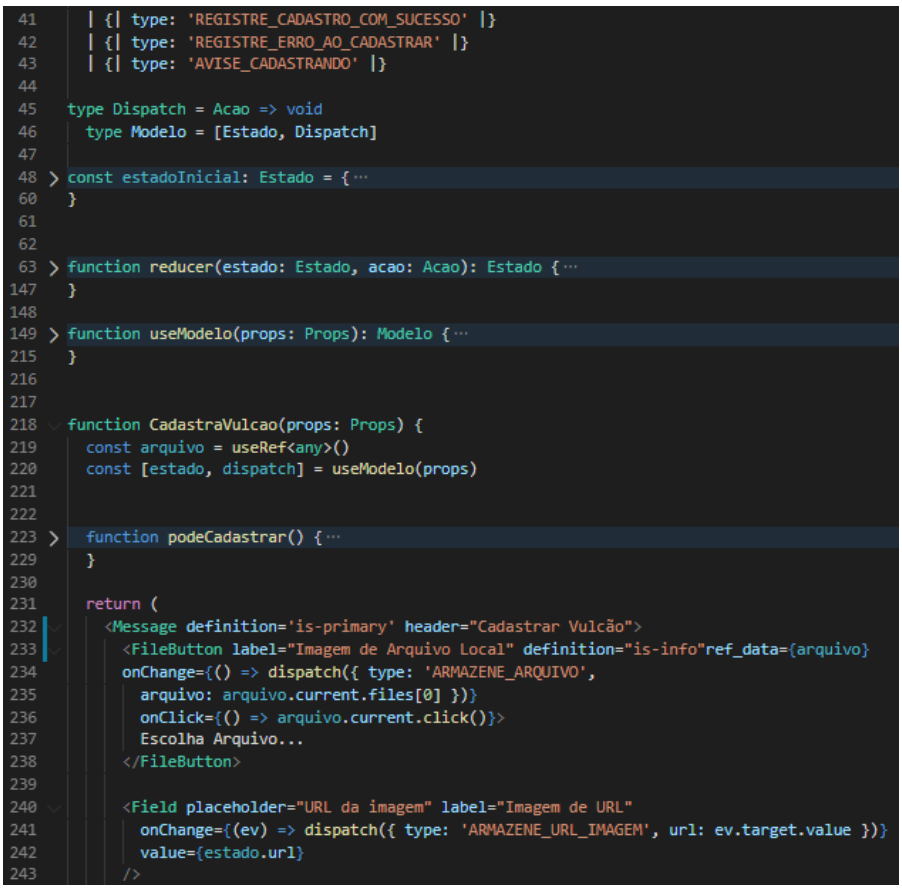
**Figura 94** - Código refatorado componente CadastraVulcao I



```
// @flow
import { Button, Field, FieldContent, FileButton, Message } from 'assemble-react-bulma'
import React, { useEffect, useReducer, useRef } from 'react'
import { cadastraVulcao } from '../servicos'
import type { VulcaoParaCadastro } from '../tipos'
import processaImagem from '../util_jimp'
import MsgModal from './MsgModal.jsx'

type Props = {|
  token: Token,
    onTokenInvalido: () => void,
      limiteImagem: number
        |}

type Estado = {|
  arquivo: any | void,
    imagem: ImagemEmBase64 | void,
      miniatura: ImagemEmBase64 | void,
        url: ImagemEmURL,
          lendoDeURL: boolean,
            pais: string,
              nome: string,
                cadastrando: boolean,
                  msgErro: string | void,
                    msgModal: string | void,
                      erro: boolean
                        |}

type Acao =
  {| type: 'ARMAZENE_ARQUIVO', arquivo: any |}
  | {| type: 'REGISTRE_LENDO_IMAGEM' |}
  | {| type: 'REGISTRE_PROCESSANDO_IMAGEM' |}
  | {| type: 'REGISTRE_IMAGEM_LIDA', imagem: ImagemEmBase64, miniatura: ImagemEmBase64 |}
  | {| type: 'REGISTRE_IMAGEM_MUITO_GRANDE', limite: number, tamanho: number |}
  | {| type: 'REGISTRE_ERRO_AO_PROCESSAR_IMAGEM', msg: string |}
  | {| type: 'ARMAZENE_URL_IMAGEM', url: string |}
  | {| type: 'LEIA_IMAGEM_DE_URL' |}
  | {| type: 'ARMAZENE_NOME', nome: string |}
  | {| type: 'ARMAZENE_PAIS', pais: string |}
  | {| type: 'CADASTRE_VULCAO' |}
```

**Fonte**: O próprio autor, 2020

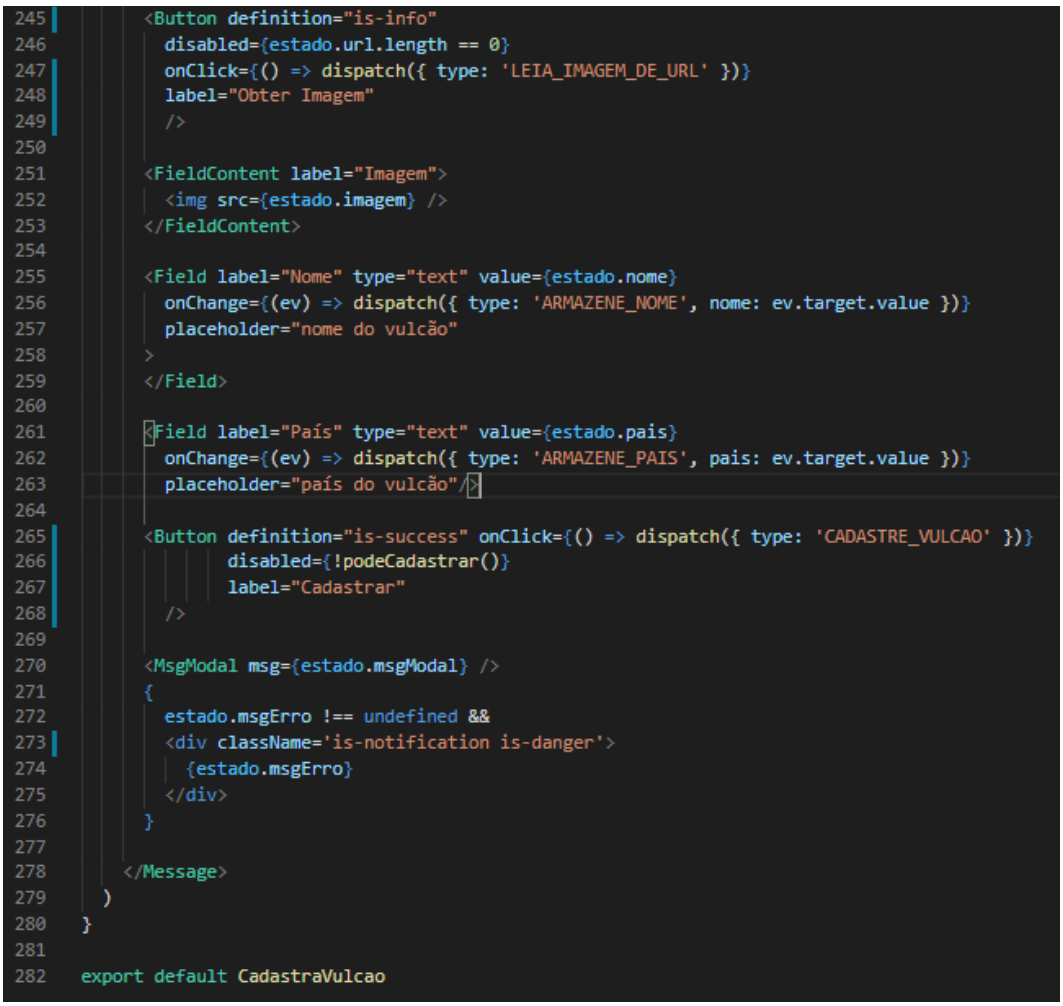
**Figura 95** - Código refatorado componente CadastraVulcao II



```
41    | {| type: 'REGISTRE_CADASTRO_COM_SUCESSO' |}
42    | {| type: 'REGISTRE_ERRO_AO_CADASTRAR' |}
43    | {| type: 'AVISE_CADASTRANDO' |}
44
45  type Dispatch = Acao => void
46    type Modelo = [Estado, Dispatch]
47
48  const estadoInicial: Estado = {···
60  }
61
62
63  function reducer(estado: Estado, acao: Acao): Estado {···
147 }
148
149 function useModelo(props: Props): Modelo {···
215 }
216
217
218 function CadastraVulcao(props: Props) {
219   const arquivo = useRef<any>()
220   const [estado, dispatch] = useModelo(props)
221
222
223   function podeCadastrar() {···
229   }
230
231   return (
232     <Message definition='is-primary' header="Cadastrar Vulcão">
233       <FileButton label="Imagem de Arquivo Local" definition="is-info"ref_data={arquivo}
234       onChange={() => dispatch({ type: 'ARMAZENE_ARQUIVO',
235         arquivo: arquivo.current.files[0] })}
236         onClick={() => arquivo.current.click()}>
237         Escolha Arquivo...
238       </FileButton>
239
240       <Field placeholder="URL da imagem" label="Imagem de URL"
241         onChange={(ev) => dispatch({ type: 'ARMAZENE_URL_IMAGEM', url: ev.target.value })}
242         value={estado.url}
243       />
```

**Fonte**: O próprio autor, 2020

**Figura 96** - Código refatorado componente CadastraVulcao III



```
245        <Button definition="is-info"
246          disabled={estado.url.length == 0}
247          onClick={() => dispatch({ type: 'LEIA_IMAGEM_DE_URL' })}
248          label="Obter Imagem"
249          />
250
251        <FieldContent label="Imagem">
252          <img src={estado.imagem} />
253        </FieldContent>
254
255        <Field label="Nome" type="text" value={estado.nome}
256          onChange={(ev) => dispatch({ type: 'ARMAZENE_NOME', nome: ev.target.value })}
257          placeholder="nome do vulcão"
258        >
259        </Field>
260
261        <Field label="País" type="text" value={estado.pais}
262          onChange={(ev) => dispatch({ type: 'ARMAZENE_PAIS', pais: ev.target.value })}
263          placeholder="país do vulcão"/>
264
265        <Button definition="is-success" onClick={() => dispatch({ type: 'CADASTRE_VULCAO' })}
266                disabled={!podeCadastrar()}
267                label="Cadastrar"
268          />
269
270        <MsgModal msg={estado.msgModal} />
271        {
272          estado.msgErro !== undefined &&
273          <div className='is-notification is-danger'>
274            {estado.msgErro}
275          </div>
276        }
277
278      </Message>
279    )
280  }
281
282  export default CadastraVulcao
```

**Fonte**: O próprio autor, 2020

Na **Figura *94***,**Figura *95*** e **Figura *96*** é possível observar que existem 72 linhas HTML (201-302), 41 tags, além de uma quantidade total de 306 linhas do código original do componente.

A primeira redução é gerada pelo componente Message (as mesmas discutidas nos tópicos 5.1.1.2 e 5.1.1.4

É possível observar que na **Figura *90***, após a tag com className “message-body”, existe um elemento com className “field”, sendo que esse elemento é pai de label, control, input do tipo “file” e button. Todo esse conjunto de elementos é codificado para criar um botão que possua tanto o evento de *onClick* quanto o de *onChange*, sendo o de *onClick* para abrir uma interface para escolha de um arquivo qualquer e o *onChange* para quando o arquivo for escolhido, realizar um determinado evento (como se mesclasse o componente File com Button). Para facilitar o trabalho

ao criar um elemento desse tipo, decidiu-se criar o componente FileButton, que proporciona um botão que abstrai toda a implementação antes mencionada, sendo esse componente utilizado no *refactoring*, conforme mostrado na **Figura *95***.

Após FileButton, adicionou-se os componentes Field, Button e FieldContent. As melhorias acarretadas por Button e Field são as mesmas explicadas no Refactoring de App Lembretes, em 5.1.1. Porém, pontua-se a utilização de FieldContent, que é uma espécie de "container" que abstrai as tags label, control e field, sendo demostrado na **Figura *96*** no seu uso, onde é passado o label como *props* e adicionado um *children* que é uma imagem de um vulcão.

O componente CadastraVulcao apresenta uma redução de 68.29% no número de tags, 34,72% no número de linhas HTML e 7,84% no número total de linhas do componente. Utilizou-se os componentes Message, FileButton, Field, FieldContent e Button, entretanto a redução de tags foi ocasionada em maior parte pelos componentes FileButton, Field e FieldContent.

#### *5.1.2.2 Métricas Refactoring*

No presente subtópico são apresentadas as métricas do refactoring aplicado nos componentes de App Vulcões.

**Tabela 10** - Legendas de App Vulcao

| Sigla | Descrição |
|---|---|
| L O | Total de linhas do código original |
| L R | Total de linhas do código refatorado |
| TO | Total de tags do código original |
| TR | Total de tags do código refatorado |
| LHO | Total de linhas HTML do código original |
| LHR | Total de linhas HTML do código refatorado |
| %LH | Porcentagem de redução do código HTML |
| %LT | Porcentagem de redução do código total |
| %T | Porcentagem de redução do número de tags |

Fonte: O próprio autor,2020

**Tabela 11** - Métricas refactoring App Vulcao

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| CadastraVulcao | 306 | 282 | 41 | 13 | 72 | 47 | 7.84 | 68.29 | 34.72 |

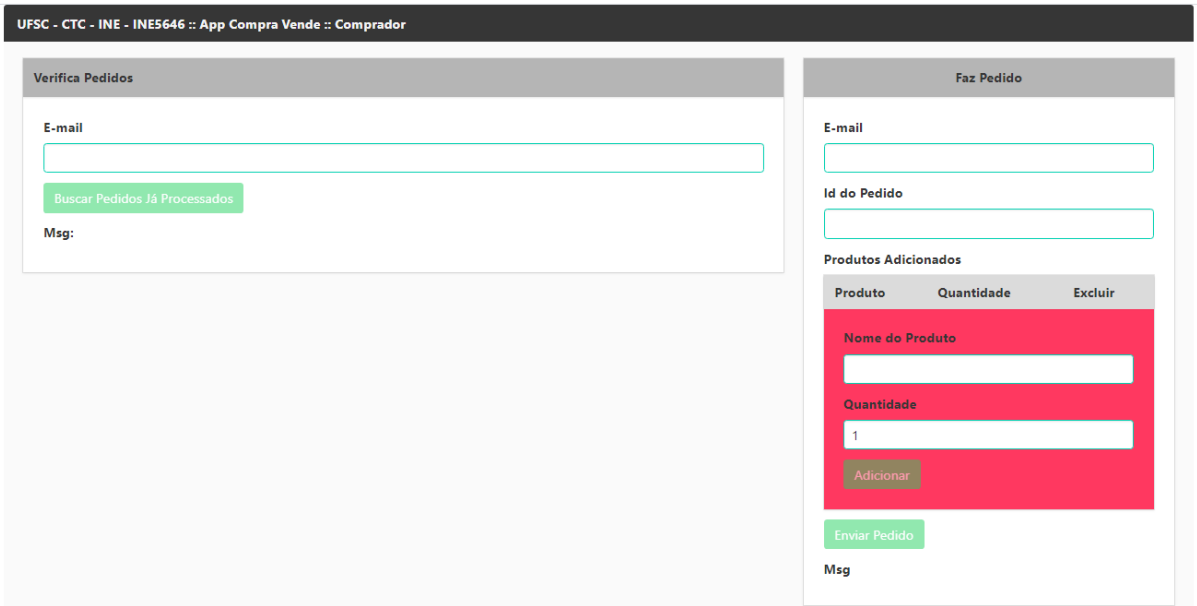
**Fonte**: O próprio autor, 2020

### 5.1.3 App Compra e Vende

Essa aplicação como um todo, trata de dois sistemas separados um para a compra de produtos e outro para confirmação de venda dos produtos. São separados em App Compra e App Venda internamente. A interface gráfica de App Compra é dividida em 6 componentes, que são: App, FazPedido, MostraPedidosProcessados, TabelaProdutos, TabelaProdutosProcessados e VerificaPedido. A interface gráfica de App Compra é dividida em 5 componentes, que são: App, ProcessaPedido, TabelaPedidos, TabelaProdutoProcessado e VerificaPedidos

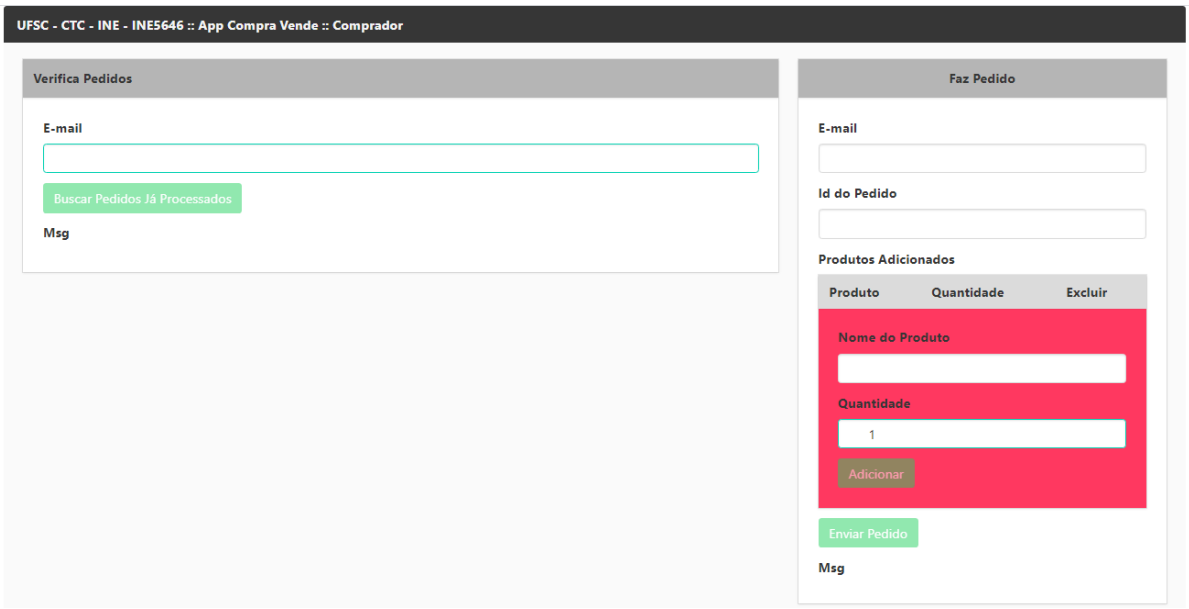
#### *5.1.3.1 App*

**Figura 97** – Imagem original do componente App de App Compra




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 98** - Imagem do componente App de App Compra pós refactoring




**Fonte**: O próprio autor, 2020

Observa-se que o resultado gráfico após o *refactoring* é o mesmo que a interface original.

**Figura 99** - Código original do componente App de App Compra.

```
// @flow
import React from 'react'
import FazPedido from './FazPedido.jsx'
import VerificaPedido from './VerificaPedido.jsx'

function App () {
  return (
    <div className="container is-fluid">
      <div className="message is-dark">
        <div className="message-header">
        UFSC - CTC - INE - INE5646 :: App Compra Vende :: Comprador
        </div>
        <div className="message-body">
          <div className='columns'>
            <div className='column is-two-thirds'><VerificaPedido /></div>
            <div className='column'><FazPedido /></div>
          </div>
        </div>
      </div>
    </div>
  )
}

export default App
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 100** - Código após refactoring do componente App de App Compra

```
// @flow
import React from 'react'
import FazPedido from './FazPedido.jsx'
import VerificaPedido from './VerificaPedido.jsx'
import {Columns,Column,Message} from 'assemble-react-bulma';

function App () {
  return (
    <div className="container is-fluid">
      <Message definition="is-dark" header="UFSC - CTC - INE - INE5646 :: App Compra Vende :: Comprador">
        <Columns>
          <Column definition="is-two-thirds"> <VerificaPedido/> </Column>
          <Column><FazPedido/> </Column>
        </Columns>
      </Message>
    </div>
  )
}

export default App
```

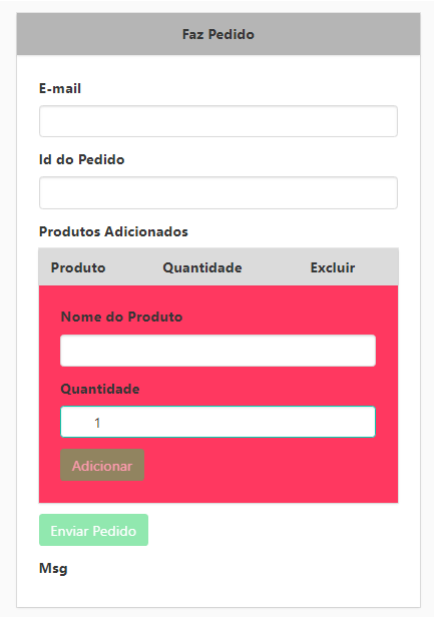
**Fonte**: Autor,2020

O componente App apresentou uma redução de 33,33% no número de tags, 38,46% no número de linhas HTML e 16,67% da quantidade total de linhas.

### *5.1.3.2 FazPedido*

Para realizar o *refactoring* de faz pedido, utilizou-se os componentes Card, FieldContent, Field e FieldButton.

**Figura 101** - Imagem original do componente FazPedido




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 102** - Imagem após refactoring do componente FazPedido


**Fonte**: O próprio autor,2020

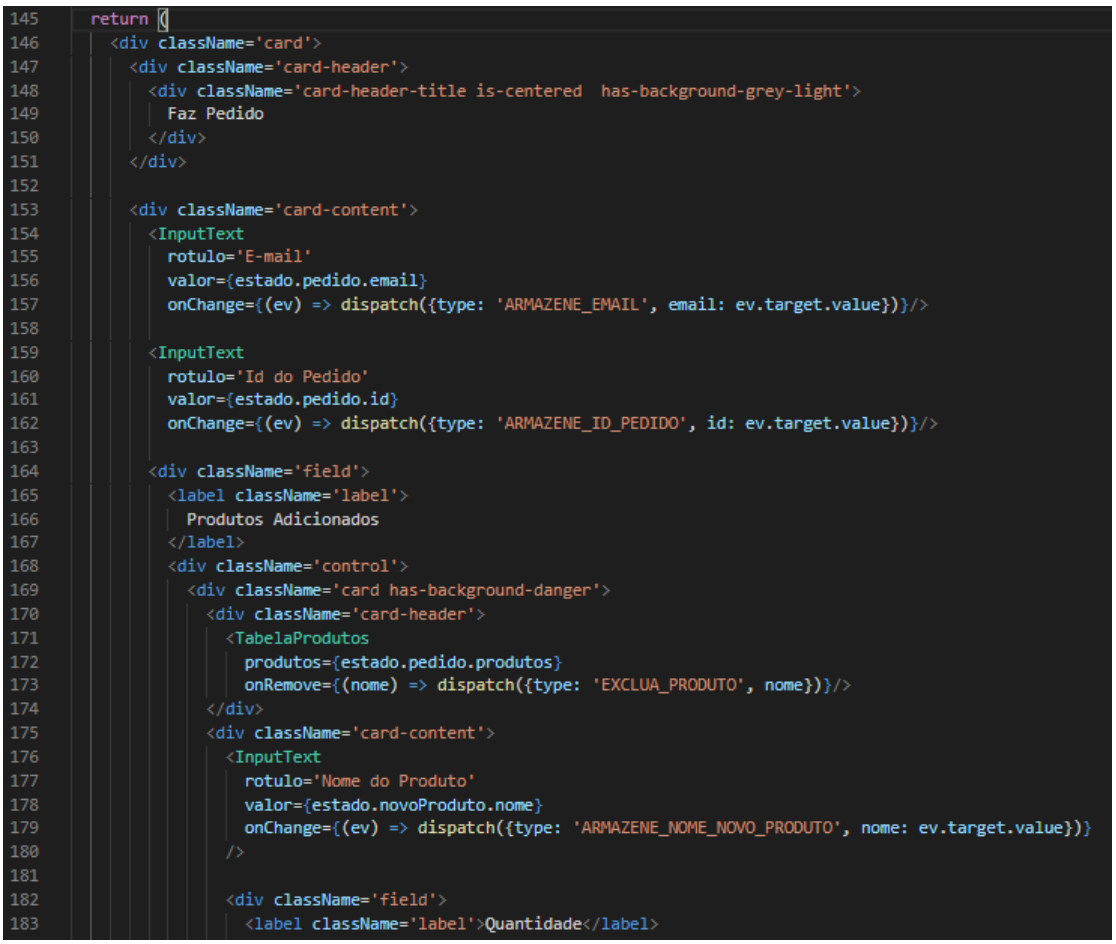
Observa-se que não ocorreram mudanças no resultado gráfico após o *refactoring* e a interface original

**Figura 103** - Código original do componente FazPedido I

```
1   //@flow
2   import React, {useReducer, useEffect} from 'react'
3
4   import {enviaPedido} from '../servicos'
5   import {Produto, Pedido} from '../modelos'
6   import {TabelaProdutos} from './TabelaProdutos.jsx'
7
8
9   type Estado = {|···
14  |}
15
16    type Acao =···
27  type Dispatch = Acao => void
28
29  type Modelo = [Estado, Dispatch]
30
31
32  const estadoInicial: Estado = {···
37  }
38
39  function reducer(estado: Estado, acao: Acao): Estado {···
123 }
124
125 function useModelo(): Modelo {···
139 }
140
141 function FazPedido () {
142   const [estado, dispatch] = useModelo()
143
144
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 104** - Código original do componente FazPedido II.



```
145   return (
146     <div className='card'>
147       <div className='card-header'>
148         <div className='card-header-title is-centered  has-background-grey-light'>
149           Faz Pedido
150         </div>
151       </div>
152
153       <div className='card-content'>
154         <InputText
155           rotulo='E-mail'
156           valor={estado.pedido.email}
157           onChange={(ev) => dispatch({type: 'ARMAZENE_EMAIL', email: ev.target.value})}/>
158
159         <InputText
160           rotulo='Id do Pedido'
161           valor={estado.pedido.id}
162           onChange={(ev) => dispatch({type: 'ARMAZENE_ID_PEDIDO', id: ev.target.value})}/>
163
164         <div className='field'>
165           <label className='label'>
166             Produtos Adicionados
167           </label>
168           <div className='control'>
169             <div className='card has-background-danger'>
170               <div className='card-header'>
171                 <TabelaProdutos
172                   produtos={estado.pedido.produtos}
173                   onRemove={(nome) => dispatch({type: 'EXCLUA_PRODUTO', nome})}/>
174               </div>
175               <div className='card-content'>
176                 <InputText
177                   rotulo='Nome do Produto'
178                   valor={estado.novoProduto.nome}
179                   onChange={(ev) => dispatch({type: 'ARMAZENE_NOME_NOVO_PRODUTO', nome: ev.target.value})}
180                 />
181
182                 <div className='field'>
183                   <label className='label'>Quantidade</label>
```

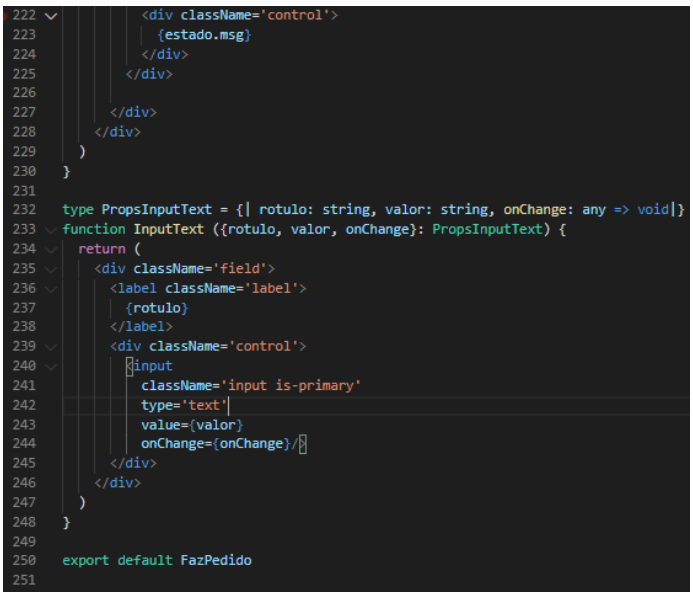
**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 105** - Código original do componente FazPedido III

```
                    <label className='label'>Quantidade</label>
                    <div className='control'>
                      <input
                        className='input is-primary'
                        type='number'
                        min='1'
                        value={estado.novoProduto.quantidade}
                        onChange={(ev) => dispatch({type: 'ARMAZENE_QTD_NOVO_PRODUTO',
                        qtdDigitada: ev.target.value})}/>
                    </div>
                  </div>
                  <div className='field'>
                    <div className='control'>
                      <button
                        disabled={estado.novoProduto.nome === ''}
                        className='button is-success'
                        onClick={() => dispatch({type: 'ADICIONE_NOVO_PRODUTO'})}>
                            Adicionar
                      </button>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>

          <div className='field'>
            <div className='control'>
              <button
                className='button is-success'
                disabled={!estado.pedido.estaPreenchido() || estado.enviando}
                onClick={() => dispatch({type: 'ENVIE_PEDIDO'})}>
                    Enviar Pedido
              </button>
            </div>
          </div>

          <div className='field'>
            <label className='label'>Msg</label>
            <div className='control'>
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

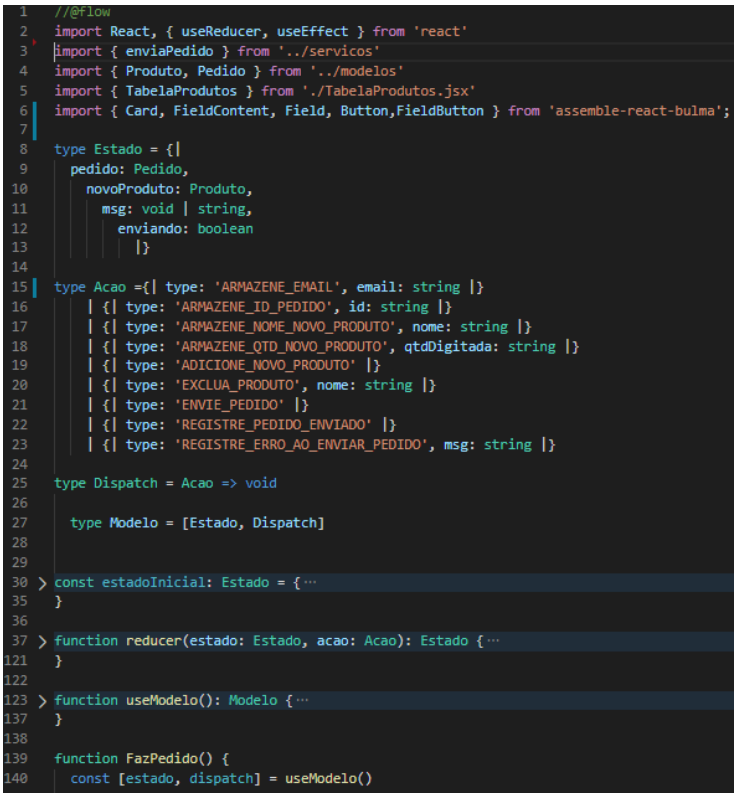
**Figura 106** - Código original do componente FazPedido IV



```
            <div className='control'>
              {estado.msg}
            </div>
          </div>

        </div>
      </div>
    )
  }

type PropsInputText = {| rotulo: string, valor: string, onChange: any => void|}
function InputText ({rotulo, valor, onChange}: PropsInputText) {
  return (
    <div className='field'>
      <label className='label'>
        {rotulo}
      </label>
      <div className='control'>
        <input
          className='input is-primary'
          type='text'
          value={valor}
          onChange={onChange}/>
      </div>
    </div>
  )
}

export default FazPedido
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

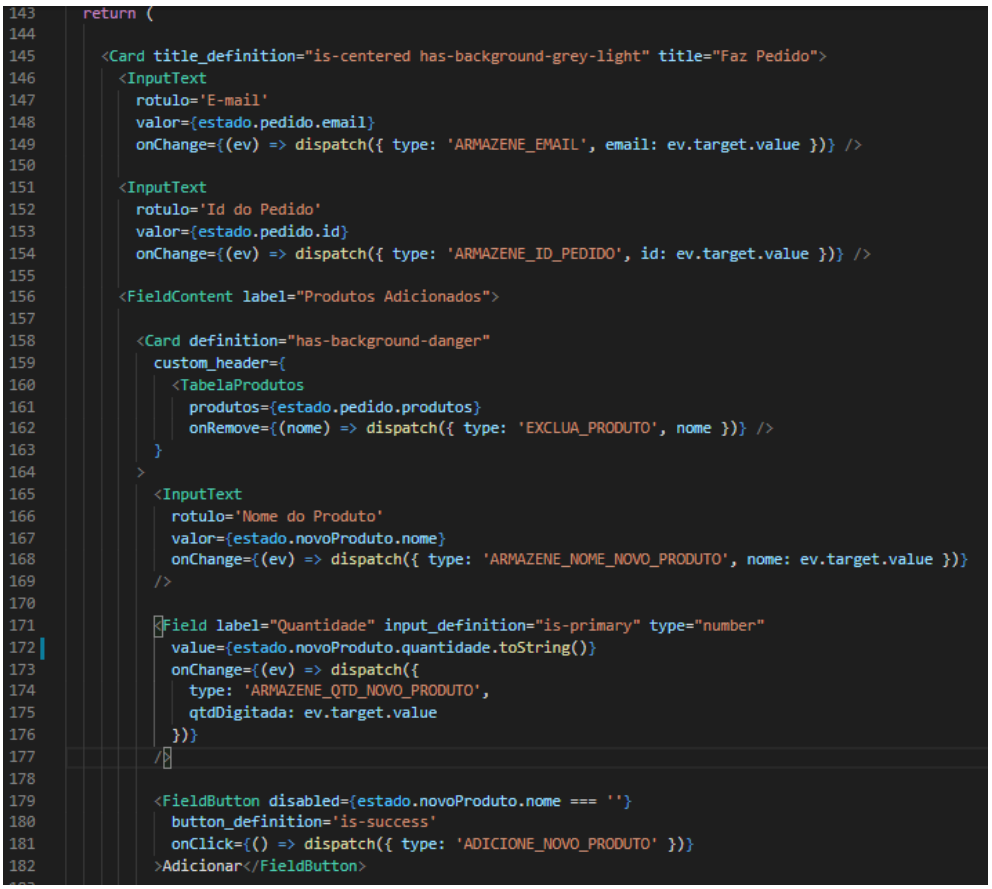
**Figura 107** - Código refactoring componente FazPedidos I



```
1   //@flow
2   import React, { useReducer, useEffect } from 'react'
3   import { enviaPedido } from '../servicos'
4   import { Produto, Pedido } from '../modelos'
5   import { TabelaProdutos } from './TabelaProdutos.jsx'
6   import { Card, FieldContent, Field, Button,FieldButton } from 'assemble-react-bulma';
7
8   type Estado = {|
9     pedido: Pedido,
10      novoProduto: Produto,
11       msg: void | string,
12        enviando: boolean
13          |}
14
15  type Acao ={| type: 'ARMAZENE_EMAIL', email: string |}
16      | {| type: 'ARMAZENE_ID_PEDIDO', id: string |}
17      | {| type: 'ARMAZENE_NOME_NOVO_PRODUTO', nome: string |}
18      | {| type: 'ARMAZENE_QTD_NOVO_PRODUTO', qtdDigitada: string |}
19      | {| type: 'ADICIONE_NOVO_PRODUTO' |}
20      | {| type: 'EXCLUA_PRODUTO', nome: string |}
21      | {| type: 'ENVIE_PEDIDO' |}
22      | {| type: 'REGISTRE_PEDIDO_ENVIADO' |}
23      | {| type: 'REGISTRE_ERRO_AO_ENVIAR_PEDIDO', msg: string |}
24
25  type Dispatch = Acao => void
26
27    type Modelo = [Estado, Dispatch]
28
29
30  const estadoInicial: Estado = {···
35  }
36
37  function reducer(estado: Estado, acao: Acao): Estado {···
121 }
122
123 function useModelo(): Modelo {···
137 }
138
139 function FazPedido() {
140   const [estado, dispatch] = useModelo()
```

**Fonte**: O próprio autor,2020

**Figura 108** - Código refactoring componente FazPedidos II



```
return (

  <Card title_definition="is-centered has-background-grey-light" title="Faz Pedido">
    <InputText
      rotulo='E-mail'
      valor={estado.pedido.email}
      onChange={(ev) => dispatch({ type: 'ARMAZENE_EMAIL', email: ev.target.value })} />

    <InputText
      rotulo='Id do Pedido'
      valor={estado.pedido.id}
      onChange={(ev) => dispatch({ type: 'ARMAZENE_ID_PEDIDO', id: ev.target.value })} />

    <FieldContent label="Produtos Adicionados">

      <Card definition="has-background-danger"
        custom_header={
          <TabelaProdutos
            produtos={estado.pedido.produtos}
            onRemove={(nome) => dispatch({ type: 'EXCLUA_PRODUTO', nome })} />
        }
      >
        <InputText
          rotulo='Nome do Produto'
          valor={estado.novoProduto.nome}
          onChange={(ev) => dispatch({ type: 'ARMAZENE_NOME_NOVO_PRODUTO', nome: ev.target.value })}
        />

        <Field label="Quantidade" input_definition="is-primary" type="number"
          value={estado.novoProduto.quantidade.toString()}
          onChange={(ev) => dispatch({
            type: 'ARMAZENE_QTD_NOVO_PRODUTO',
            qtdDigitada: ev.target.value
          })}
        />

        <FieldButton disabled={estado.novoProduto.nome === ''}
          button_definition='is-success'
          onClick={() => dispatch({ type: 'ADICIONE_NOVO_PRODUTO' })}
        >Adicionar</FieldButton>
```

**Fonte**: O próprio autor,2020

**Figura 109** - Código refactoring componente FazPedidos III

```
183
184           </Card>
185         </FieldContent>
186         <FieldButton
187           button_definition='is-success'
188           disabled={!estado.pedido.estaPreenchido() || estado.enviando}
189           onClick={() => dispatch({ type: 'ENVIE_PEDIDO' })} >Enviar Pedido </FieldButton>
190
191         <FieldContent label="Msg">
192           {estado.msg}
193         </FieldContent>
194
195       </Card>
196     )
197   }
198
199   type PropsInputText = {| rotulo: string, valor: string, onChange: any => void|}
200   function InputText({ rotulo, valor, onChange }: PropsInputText) {
201     return (
202       <Field label={rotulo} type="text"
203         value={valor.toString()}
204         onChange={onChange}
205       />
206     )
207   }
208
209   export default FazPedido
```

**Fonte**: O próprio autor,2020

Ao utilizar Card é excluída a necessidade de informar as tags “card”, “card-header”, “card-header-title” e “card-content”, pois o componente Card trata a sua implementação internamente criando essas tags conforme as informações fornecidas via *props*. A tag “card” é o elemento principal do componente, a tag “card-header” é formada a partir do atributo title(*props*), logo não precisou ser informada, a tag “card-header-title” é criada internamente a partir das propriedades title(*props*) e title_definition (*props*) e a tag “card-content” é equivalente a *children* de Card. Nota-se uma considerável redução de código gerada pelo componente Card nessa situação.

Após Card, o componente interno InputText é invocado, o código refatorado de InputText é apresentado na **Figura 109**.

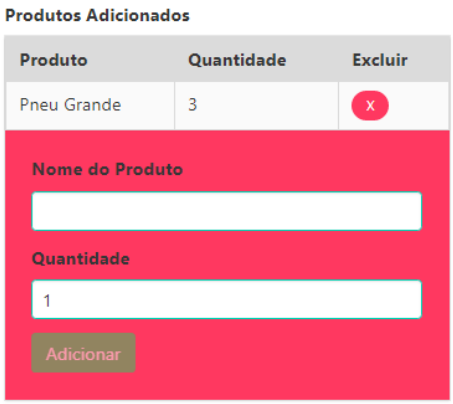
Após a invocação de InputText é chamado FieldContent e depois invocado outro componente Card. Observa-se nesse componente o atributo custom_header, que foi criado justamente para possibilitar uma customização do Header já que o “header default” permite somente texto e em alguns casos é interessante customizar, portanto definiu-se essa possibilidade.

O código após o *refactoring* de FazPedido revela um componente chamado FieldButton. Esse componente foi criado a partir da observação de que algumas vezes nas aplicações, o componente Button era renderizado “dentro” de uma estrutura de “field” e essa estrutura necessitava de uma considerável quantidade de código, conforme é possível observar na **Figura *105*** onde o código de “Enviar Pedido” é apresentado. Ao criar e utilizar FieldButton foi possível simplificar a utilização dessa estrutura conforme apresentado na **Figura 109** com o *refactoring.*

O componente FazPedido apresentou uma redução de 70,37% do número de tags, redução de 51,75% no número de linhas HTML e 15,32% do número total de linhas.

### *5.1.3.3 TabelaProdutos*

**Figura 110** - Imagem original do componente TabelaProdutos




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 111** - Imagem após refactoring do componente TabelaProdutos


**Fonte**: O próprio autor,2020

**Figura 112** - Código original componente TabelaProdutos I

```
//@flow
import React from 'react'
import {Produto} from '../modelos'

type Props = {|
  produtos: Array<Produto>,
  onRemove: string => void
|}

const TabelaProdutos = (props: Props) => {

  let produtos = props.produtos.map(p =>
    <tr key={p.nome}>
      <td>{p.nome}</td>
      <td>{p.quantidade}</td>
      <td>
        <button className='button is-danger is-small is-rounded'
          onClick={() => props.onRemove(p.nome)}>X</button>
      </td>
    </tr>)

  return (
    <table className='table is-striped is-hoverable is-bordered is-fullwidth'>
      <thead className='has-background-grey-lighter'>
        <tr>
          <th>Produto</th>
          <th>Quantidade</th>
          <th>Excluir</th>
        </tr>
      </thead>
      <tbody>
        {produtos}
      </tbody>
    </table>
  )
}

export {TabelaProdutos}
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 113** - Código após refactoring do componente TabelaProdutos

```
// @flow
import React from 'react'
import { Produto } from '../modelos'
import { Table, Button } from 'assemble-react-bulma';

type Props = {|
  produtos: Array < Produto >,
    onRemove: string => void
|}

const TabelaProdutos = (props: Props) => {
  var itens_header = [{ value: "Produto" }, { value: "Quantidade" }, { value: "Excluir" }];

  var itens_body = props.produtos.map(elemento => ({
      nome: elemento.nome,
     quantidade: elemento.quantidade,
     botao: <Button definition='is-danger is-small is-rounded'
                    label="X"
                    onClick={() => props.onRemove(elemento.nome)}/>
  }))

  return (
   <Table definition='is-striped is-hoverable is-bordered is-fullwidth'  header_definition="has-background-grey-lighter"
          itens_header={itens_header} itens_body={itens_body}/>
  )
}

export { TabelaProdutos }
```

**Fonte**: O próprio autor,2020

No código original de TabelaProdutos, na **Figura 112**, é possível observar uma significativa utilização de tags, como <td>, <th>, <tr>, <thead>, <tbody> e <table>. É notável a necessidade de uma grande repetição de tags.

Para o refactoring, utilizou-se o componente Table da presente solução. Esse componente fornece uma estrutura necessária para os dados que serão exibidos através dos atributos itens_header, itens_body e itens_footer de *props*, logo foi necessário converter os dados em *props.produto*s e armazená-los na variável itens_body para que a exibição correta fosse efetuada. A variável itens-header foi criada para guardar os elementos do header da aplicação.

Com o componente Table é possível notar uma grande redução do código HTML gerado, conforme apresentado na **Figura 113**, o usuário da solução precisa apenas definir a estrutura correta para header, body e footer e informá-la ao componente que ele criará o resultado gráfico, não precisando se preocupar com todas as outras tags envolvidas na construção, apenas com os dados que são desejáveis na exibição. Pontua-se que ao de desenvolver o componente Table, o objetivo era diminuir drasticamente a necessidade de lembrar tags, digitá-las e corrigi-las para se criar um elemento, observando que o tempo para isso pode ser consideravelmente grande.

O componente TabelaProdutos apresentou uma redução de 91.67% no número de tags, redução de 80.00% no número de linhas HTML e 28.95% no número total de linhas. Observa-se que para esse componente utilizou-se apenas Table e mesmo assim atingiu um número maior do que casos onde usou-se apenas um componente

#### *5.1.3.4 TabelaProdutosProcessados*

No componente TabelaProdutosProcessados, utilizou-se o componente TableQuery da presente solução para fazer o *refactoring*.

**Figura 114** - Imagem original componente TabelaProdutosProcessados

| Produto | Quantidade | Preço Unitário (em R$) | Preço (em R$) |
|---|---|---|---|
| Pneu Grande | 1 | 50 | 50 |

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 115** – Imagem após refactoring componente TabelaProdutosProcessados

| Produto | Quantidade | Preço Unitário em (R$) | Preço (em R$) |
|---|---|---|---|
| Pneu Grande | 1 | 50 | 50 |

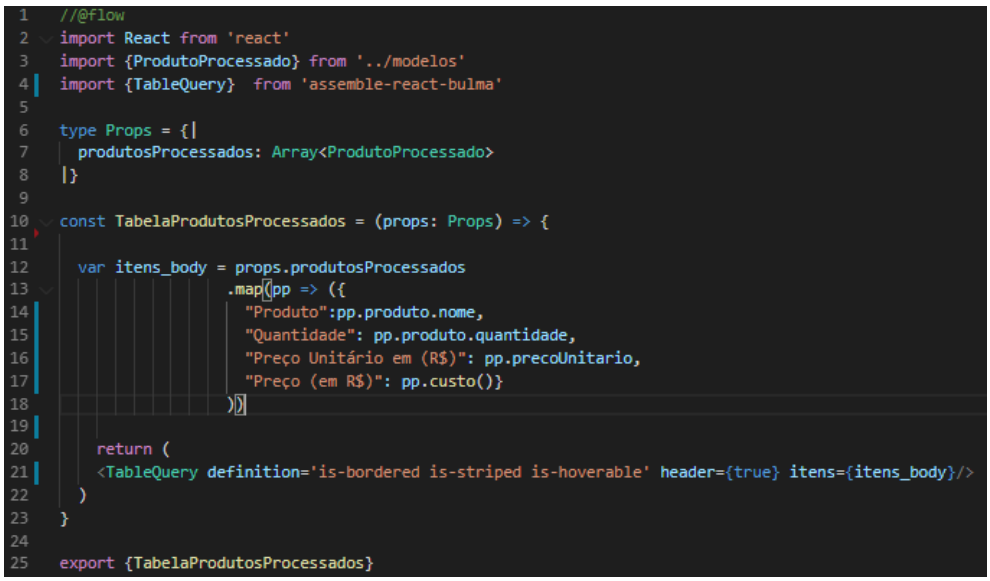
**Fonte**: O próprio autor, 2020

**Figura 116** - Código original componente TabelaProdutosProcessados

```
//@flow
import React from 'react'
import {ProdutoProcessado} from '../modelos'

type Props = {|
  produtosProcessados: Array<ProdutoProcessado>
|}

const TabelaProdutosProcessados = (props: Props) => {
  return (
    <table className='table is-bordered is-striped is-hoverable'>
      <thead>
        <tr>
          <th>Produto</th>
          <th>Quantidade</th>
          <th>Preço Unitário (em R$)</th>
          <th>Preço (em R$)</th>
        </tr>
      </thead>
      <tbody>
        {props.produtosProcessados.map(pp =>
          <tr key={pp.produto.nome}>
            <td>{pp.produto.nome}</td>
            <td>{pp.produto.quantidade}</td>
            <td>{pp.precoUnitario}</td>
            <td>{pp.custo()}</td>
          </tr>
        )}
      </tbody>
    </table>
  )
}

export {TabelaProdutosProcessados}
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 117** - Código após refactoring componente TabelaProdutosProcessados



```
//@flow
import React from 'react'
import {ProdutoProcessado} from '../modelos'
import {TableQuery}  from 'assemble-react-bulma'

type Props = {|
  produtosProcessados: Array<ProdutoProcessado>
|}

const TabelaProdutosProcessados = (props: Props) => {

  var itens_body = props.produtosProcessados
                    .map(pp => ({
                      "Produto":pp.produto.nome,
                      "Quantidade": pp.produto.quantidade,
                      "Preço Unitário em (R$)": pp.precoUnitario,
                      "Preço (em R$)": pp.custo()}
                    ))

    return (
    <TableQuery definition='is-bordered is-striped is-hoverable' header={true} itens={itens_body}/>
  )
}

export {TabelaProdutosProcessados}
```

**Fonte**: O próprio autor, 2020

O *refactoring* de TabelaProdutosProcessados poderia ser realizado com o componente Table, igual feito com o componente TabelaProdutos, entretanto optou-se pela escolha de TableQuery para pontuar algumas ideias pertinentes.

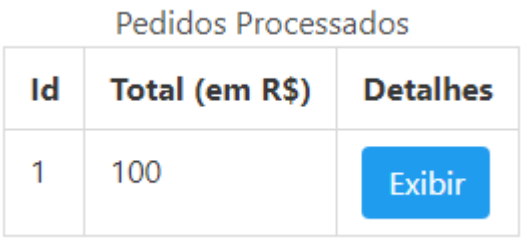
O componente TableQuery foi criado com o intuito de facilitar o uso de tabelas HTML com consultas realizadas em banco, sendo especificamente criadas apenas para exibição de dados. A ideia geral de TableQuery é possuir uma variável que seja um array de objetos e passar para TableQuery através de itens(*props*) e com isso montar uma tabela que será exibida ao usuário. Apesar do resultado da consulta ter sofrido modificações (**Figura 117**), usou-se TableQuery para pontuar as diferenças de uso em relação a Table.

No caso de Table é preciso informar o array dos elementos de header e footer caso deseje-se que sejam exibidos, no caso de TableQuery basta passar o atributo itens e caso deseje-se header ou footer, deve-se informar as propriedade footer ou header com valor true, como é feito na **Figura 117** (header={true}). Ao informar header={true} ou footer={true}, o header e footer serão compostos dos valores das chaves dos objetos que pertencem a itens(props), no caso da **Figura 117** esses valores são “Produto”, “Quantidade”, “Preço Unitário em (R$)” e “Preço (em R$).

O componente TabelaProdutosProcessados apresentou uma redução de 96,15% no número de tags, redução de 94,74% no número de linhas HTML e 26,47% no número total de linhas do componente.

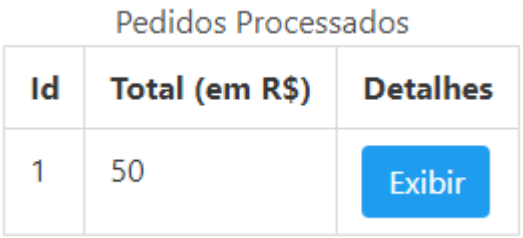
#### *5.1.3.5 MostraPedidosProcessados*

**Figura 118** - Imagem original do componente MostraPedidosProcessados




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 119** - Imagem após refactoring MostraPedidosProcessados




**Fonte**: O próprio autor,2020

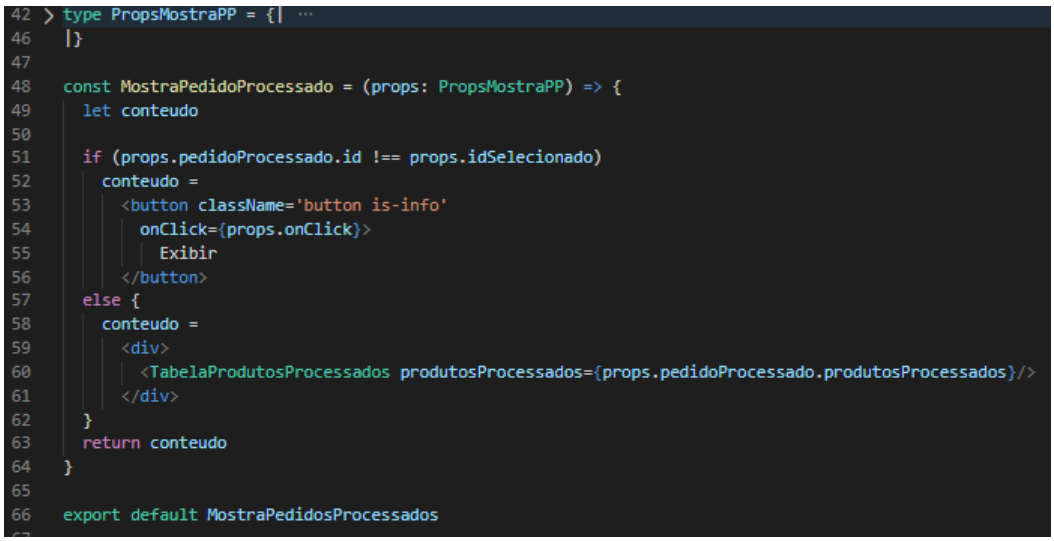
Observa-se que o resulta gráfico após o *refactoring* é igual ao apresentado no componente original

**Figura 120** - Código original MostraPedidosProcessados I.

```
//@flow
import React, {useState} from 'react'
import {TabelaProdutosProcessados} from './TabelaProdutosProcessados.jsx'
import {PedidoProcessado} from '../modelos'

type Props = {| pedidosProcessados: Array<PedidoProcessado> |}

function MostraPedidosProcessados (props: Props) {
  const [idSelecionado, setIdSelecionado] = useState(undefined)

  if (props.pedidosProcessados.length === 0) return null

  return (
    <div align='center'>
      <h3>Pedidos Processados</h3>
      <table className='table is-hoverable is-bordered is-striped'>
        <thead>
          <tr>
            <th>Id</th>
            <th>Total (em R$)</th>
            <th>Detalhes</th>
          </tr>
        </thead>
        <tbody>
          {props.pedidosProcessados.map(pp =>
            <tr key={pp.id}>
              <td>{pp.id}</td>
              <td>{pp.custo()}</td>
              <td><MostraPedidoProcessado
                pedidoProcessado={pp}
                idSelecionado={idSelecionado}
                onClick={() => setIdSelecionado(pp.id)}/>
              </td>
            </tr>
          )}
        </tbody>
      </table>
    </div>
  )
}
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 121** - Código original MostraPedidosProcessados II.

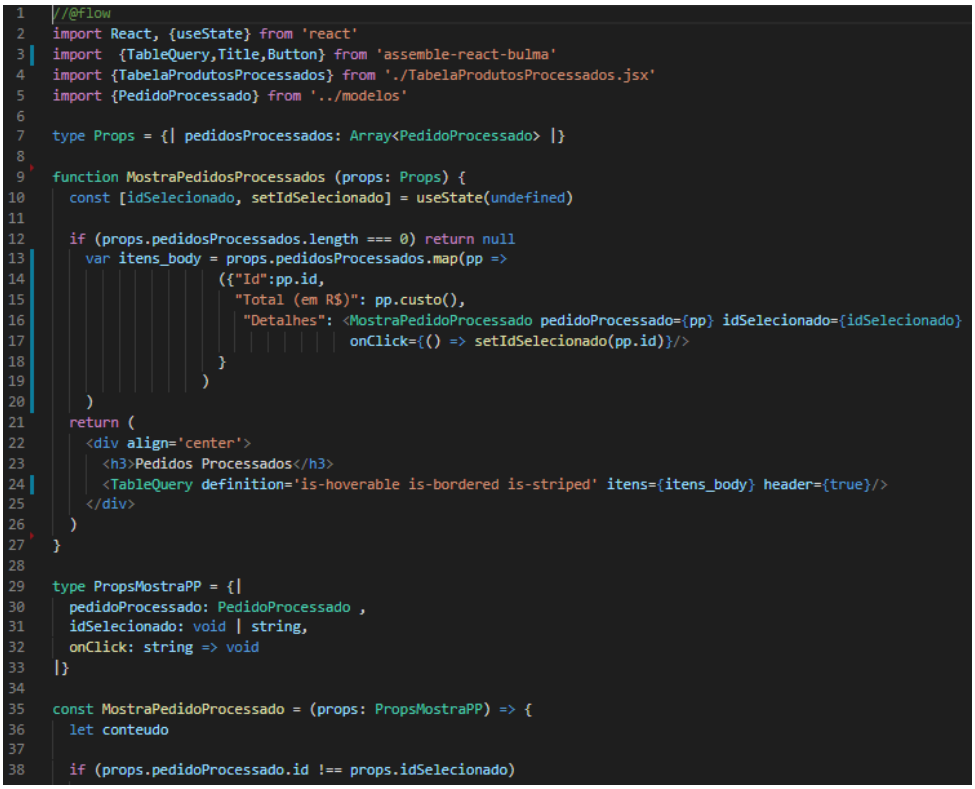


```
42 > type PropsMostraPP = {| ···
46   |}
47
48   const MostraPedidoProcessado = (props: PropsMostraPP) => {
49     let conteudo
50
51     if (props.pedidoProcessado.id !== props.idSelecionado)
52       conteudo =
53         <button className='button is-info'
54           onClick={props.onClick}>
55             Exibir
56         </button>
57     else {
58       conteudo =
59         <div>
60           <TabelaProdutosProcessados produtosProcessados={props.pedidoProcessado.produtosProcessados}/>
61         </div>
62     }
63     return conteudo
64   }
65
66   export default MostraPedidosProcessados
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na **Figura *120*** e **Figura *121*** pode-se observar que existem 32 linhas referente ao código HTML (14-38,53-56 e 59-61), 32 tags, além do código original possuir um total de 66 linhas.

**Figura 122** - Código refatorado MostraPedidosProcessados I



```
1  //@flow
2  import React, {useState} from 'react'
3  import  {TableQuery,Title,Button} from 'assemble-react-bulma'
4  import {TabelaProdutosProcessados} from './TabelaProdutosProcessados.jsx'
5  import {PedidoProcessado} from '../modelos'
6
7  type Props = {| pedidosProcessados: Array<PedidoProcessado> |}
8
9  function MostraPedidosProcessados (props: Props) {
10   const [idSelecionado, setIdSelecionado] = useState(undefined)
11
12   if (props.pedidosProcessados.length === 0) return null
13     var itens_body = props.pedidosProcessados.map(pp =>
14                    ({"Id":pp.id,
15                      "Total (em R$)": pp.custo(),
16                      "Detalhes": <MostraPedidoProcessado pedidoProcessado={pp} idSelecionado={idSelecionado}
17                                    onClick={() => setIdSelecionado(pp.id)}/>
18                    }
19                  )
20     )
21   return (
22     <div align='center'>
23       <h3>Pedidos Processados</h3>
24       <TableQuery definition='is-hoverable is-bordered is-striped' itens={itens_body} header={true}/>
25     </div>
26   )
27 }
28
29 type PropsMostraPP = {|
30   pedidoProcessado: PedidoProcessado ,
31   idSelecionado: void | string,
32   onClick: string => void
33 |}
34
35 const MostraPedidoProcessado = (props: PropsMostraPP) => {
36   let conteudo
37
38   if (props.pedidoProcessado.id !== props.idSelecionado)
```

**Fonte**: O próprio autor,2020

**Figura 123** - Código refatorado MostraPedidosProcessados II

```
39      conteudo =
40        <Button definition='is-info' onClick={props.onClick}
41          label="Exibir"
42        />
43    else {
44      conteudo =
45        <div>
46          <TabelaProdutosProcessados produtosProcessados={props.pedidoProcessado.produtosProcessados}/>
47        </div>
48    }
49    return conteudo
50  }
51
52  export default MostraPedidosProcessados
53
```

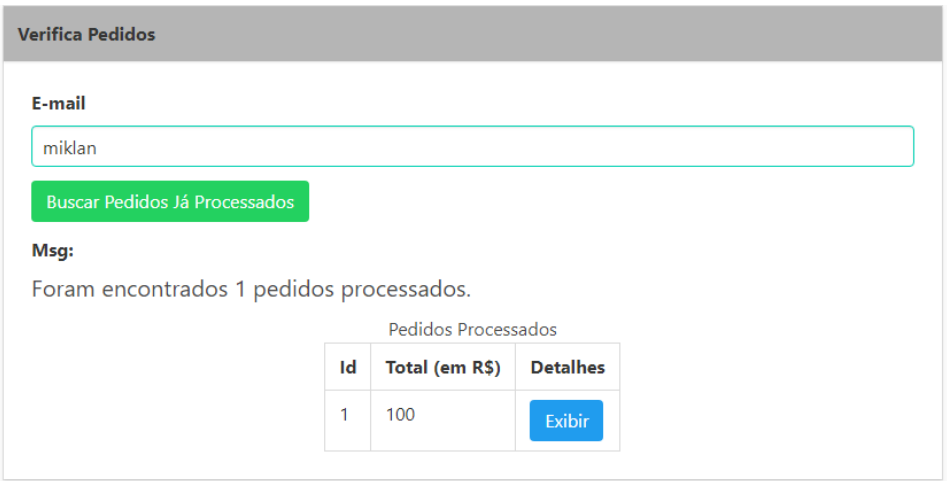
**Fonte**: O próprio autor,2020

Na Figura 122 e Figura 123, pode-se observar que existem 10 linhas referente ao código HTML (22-25,40-42 e 45-47), 10 tags, além do código refatorado possuir um total de 52 linhas.

O componente MostraPedidosProcessados apresentou uma redução de 68.75% no número de tags, redução de 68,75% no número de linhas HTML e 21,21% no número total de linhas do componente.

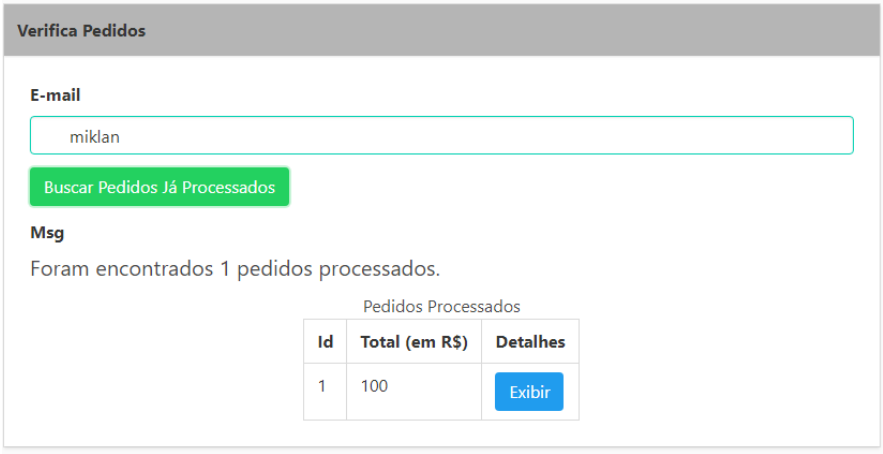
### *5.1.3.6 VerificaPedido*

**Figura 124** - Imagem Original VerificaPedido




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 125** - Imagem após refactoring VerificaPedido




**Fonte**: O próprio autor,2020

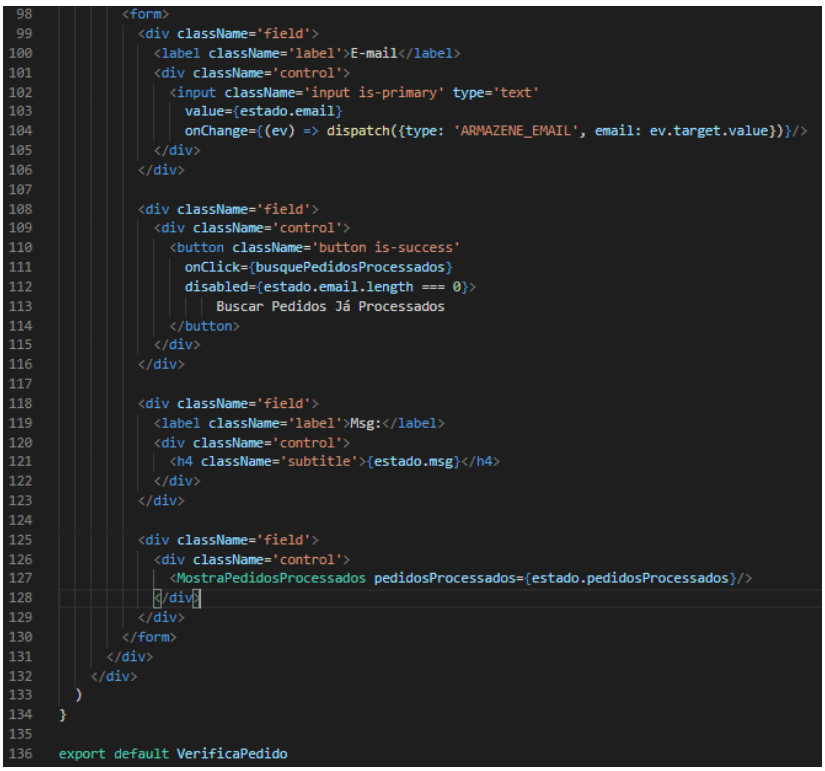
Observa-se que o resultado gráfico após o refactoring é igual a interface gráfica original do componente.

**Figura 126** - Código original VerificaPedido I

```
1   //@flow
2   import React, {useReducer, useEffect} from 'react'
3   import {buscaPedidosProcessados} from '../servicos'
4   import {PedidoProcessado} from '../modelos'
5   import MostraPedidosProcessados from './MostraPedidosProcessados.jsx'
6
7   type Estado = {|
12  |}
13
14  type Acao =
21  type Dispatch = Acao => void
22
23  type Modelo = [Estado, Dispatch]
24
25
26  const estadoInicial: Estado = {
31  }
32
33  function reducer(estado: Estado, acao: Acao): Estado {
50  }
51
52  function useModelo(): Modelo {
80  }
81
82  function VerificaPedido () {
83    const [estado, dispatch]: Modelo = useModelo()
84
85    function busquePedidosProcessados(ev) {
86      ev.preventDefault() // para não enviar requisição ao servidor
87      dispatch({type: 'REGISTRE_BUSCANDO_PEDIDOS_PROCESSADOS'})
88    }
89
90    return (
91      <div className='card'>
92        <div className= 'card-header'>
93          <div className='card-header-title has-background-grey-light'>
94                Verifica Pedidos
95          </div>
96        </div>
97        <div className='card-content'>
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 127** - Código original componente VerificaPedido II

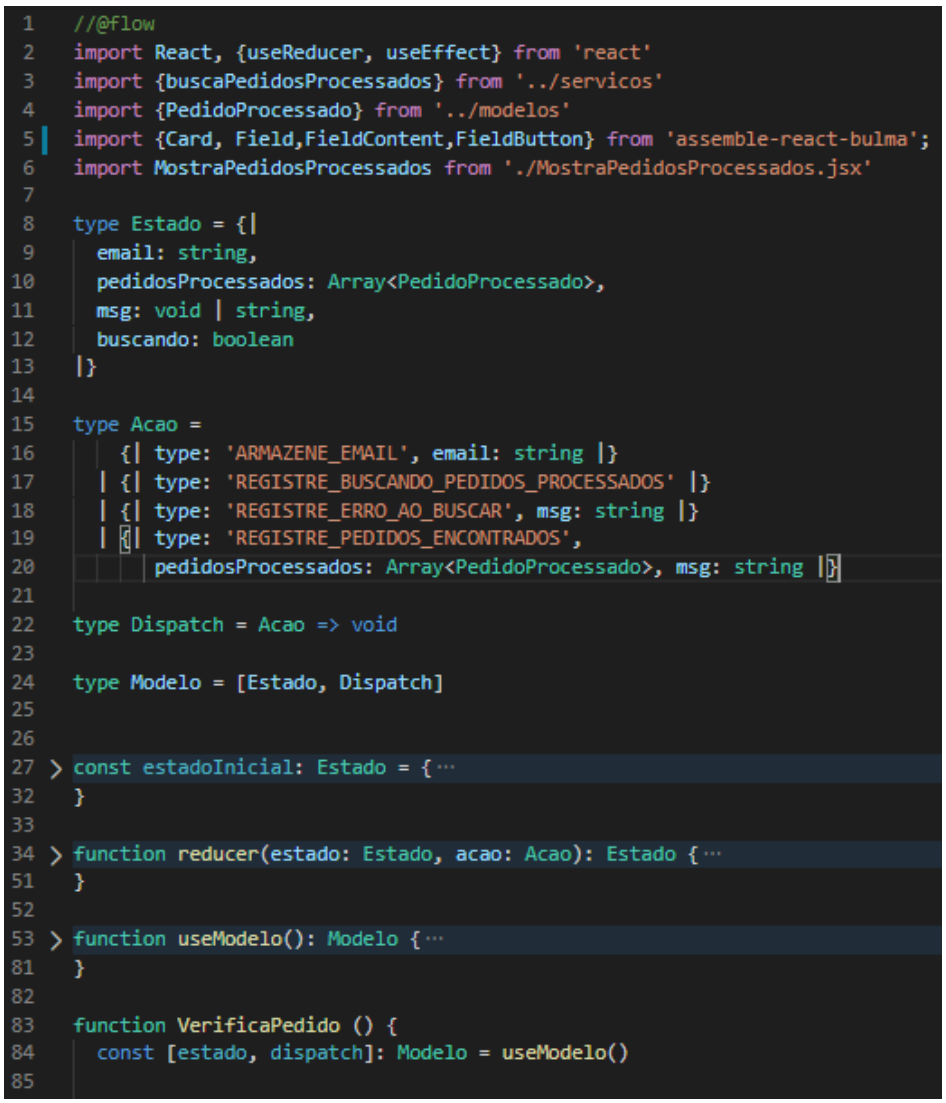


```
98            <form>
99              <div className='field'>
100               <label className='label'>E-mail</label>
101               <div className='control'>
102                 <input className='input is-primary' type='text'
103                   value={estado.email}
104                   onChange={(ev) => dispatch({type: 'ARMAZENE_EMAIL', email: ev.target.value})}/>
105               </div>
106             </div>
107
108             <div className='field'>
109               <div className='control'>
110                 <button className='button is-success'
111                   onClick={busquePedidosProcessados}
112                   disabled={estado.email.length === 0}>
113                       Buscar Pedidos Já Processados
114                 </button>
115               </div>
116             </div>
117
118             <div className='field'>
119               <label className='label'>Msg:</label>
120               <div className='control'>
121                 <h4 className='subtitle'>{estado.msg}</h4>
122               </div>
123             </div>
124
125             <div className='field'>
126               <div className='control'>
127                 <MostraPedidosProcessados pedidosProcessados={estado.pedidosProcessados}/>
128               </div>
129             </div>
130           </form>
131         </div>
132       </div>
133     )
134   }
135
136   export default VerificaPedido
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na **Figura *126*** e **Figura *127***, pode-se observar que existem 41 linhas referente ao código HTML (91-132), 36 tags, além do código original possuir um total de 136 linhas.

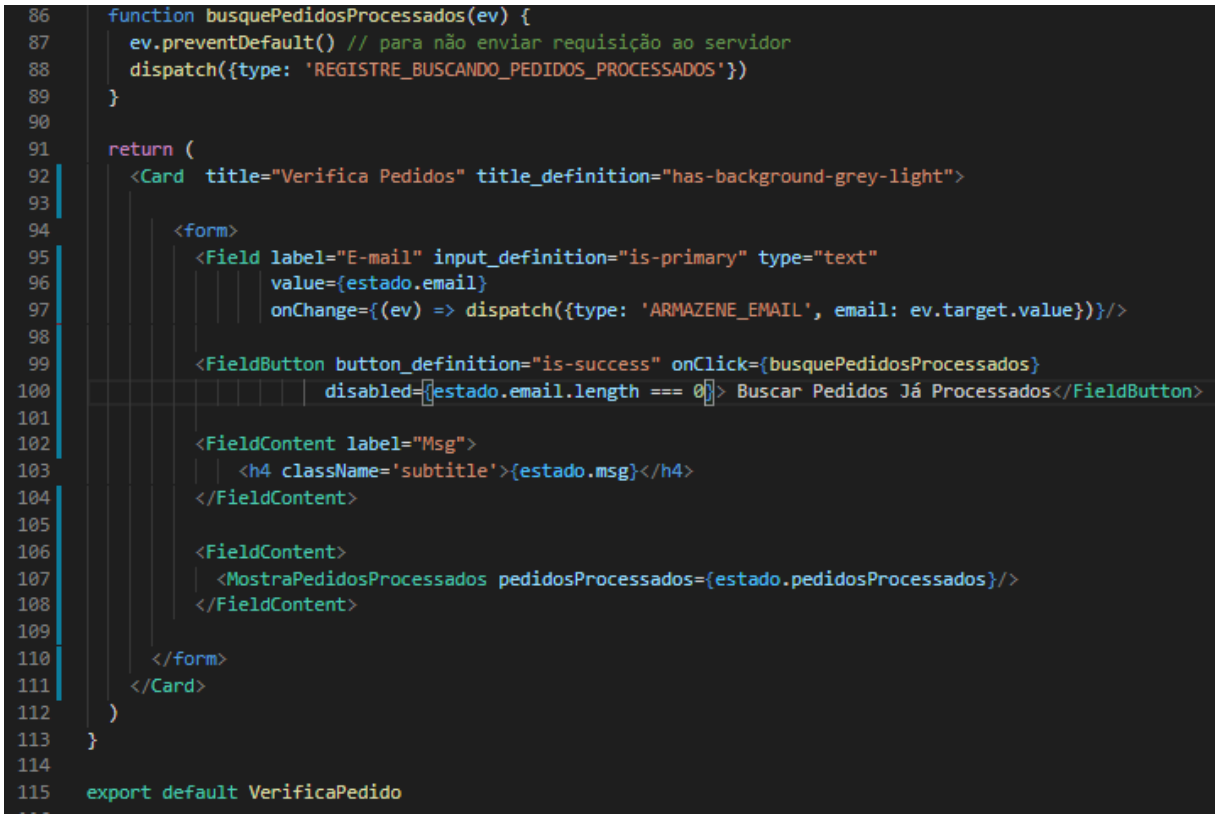
**Figura 128** - Código refatorado VerificaPedido I



```
1   //@flow
2   import React, {useReducer, useEffect} from 'react'
3   import {buscaPedidosProcessados} from '../servicos'
4   import {PedidoProcessado} from '../modelos'
5   import {Card, Field,FieldContent,FieldButton} from 'assemble-react-bulma';
6   import MostraPedidosProcessados from './MostraPedidosProcessados.jsx'
7
8   type Estado = {|
9     email: string,
10    pedidosProcessados: Array<PedidoProcessado>,
11    msg: void | string,
12    buscando: boolean
13  |}
14
15  type Acao =
16      {| type: 'ARMAZENE_EMAIL', email: string |}
17    | {| type: 'REGISTRE_BUSCANDO_PEDIDOS_PROCESSADOS' |}
18    | {| type: 'REGISTRE_ERRO_AO_BUSCAR', msg: string |}
19    | {| type: 'REGISTRE_PEDIDOS_ENCONTRADOS',
20          pedidosProcessados: Array<PedidoProcessado>, msg: string |}
21
22  type Dispatch = Acao => void
23
24  type Modelo = [Estado, Dispatch]
25
26
27  const estadoInicial: Estado = {···
32  }
33
34  function reducer(estado: Estado, acao: Acao): Estado {···
51  }
52
53  function useModelo(): Modelo {···
81  }
82
83  function VerificaPedido () {
84    const [estado, dispatch]: Modelo = useModelo()
85
```

**Fonte**: O próprio autor,2020

**Figura 129** - Código refatorado VerificaPedido II.



```
86  function busquePedidosProcessados(ev) {
87    ev.preventDefault() // para não enviar requisição ao servidor
88    dispatch({type: 'REGISTRE_BUSCANDO_PEDIDOS_PROCESSADOS'})
89  }
90
91  return (
92    <Card  title="Verifica Pedidos" title_definition="has-background-grey-light">
93
94        <form>
95          <Field label="E-mail" input_definition="is-primary" type="text"
96                 value={estado.email}
97                 onChange={(ev) => dispatch({type: 'ARMAZENE_EMAIL', email: ev.target.value})}/>
98
99          <FieldButton button_definition="is-success" onClick={busquePedidosProcessados}
100                       disabled={estado.email.length === 0}> Buscar Pedidos Já Processados</FieldButton>
101
102         <FieldContent label="Msg">
103           <h4 className='subtitle'>{estado.msg}</h4>
104         </FieldContent>
105
106         <FieldContent>
107           <MostraPedidosProcessados pedidosProcessados={estado.pedidosProcessados}/>
108         </FieldContent>
109
110       </form>
111   </Card>
112 )
113 }
114
115 export default VerificaPedido
```

**Fonte**: O próprio autor,2020

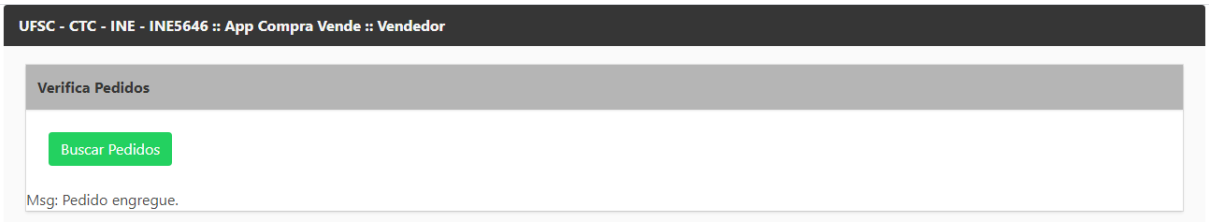
Na Figura 128 e Figura 129, pode-se observar que existem 20 linhas referente ao código HTML (92-111), 13 tags, além do código refatorado possuir um total de 115 linhas.

O componente VerificaPedido apresentou uma redução de 63.89 % no número de tags, redução de 51.22 % no número de linhas HTML e 15.44 % no número total de linhas do componente.

### *5.1.3.7 App (App Venda)*

No *refactoring* de App (App Venda) utilizou-se os componentes Message, Columns e Column.

**Figura 130** - Imagem original do componente App (App Venda)




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 131** - Imagem após refactoring do componente App (App Venda)


**Fonte**: O próprio autor,2020

**Figura 132**- Código original componente App (App Venda)

```
1  //@flow
2  import React from 'react'
3  import VerificaPedidos from './VerificaPedidos.jsx'
4  import 'bulma/css/bulma.min.css'
5
6  function App () {
7    return (
8      <div className='container is-fluid'>
9        <div className="message is-dark">
10         <div className="message-header">
11           UFSC - CTC - INE - INE5646 :: App Compra Vende :: Vendedor
12         </div>
13         <div className="message-body">
14           <div className='columns'>
15             <div className='column'>
16               <VerificaPedidos/>
17             </div>
18           </div>
19         </div>
20       </div>
21     </div>
22   )
23 }
24
25 export default App
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na Figura 132, pode-se observar que existem 14 linhas referente ao código HTML (8-21), 13 tags, além do código original possuir um total de 25 linhas.

**Figura 133** - Código refatorado componente App (App Venda)

```
1  //@flow
2  import React from 'react'
3  import VerificaPedidos from './VerificaPedidos.jsx'
4  import 'bulma/css/bulma.min.css'
5  import {Message,Columns,Column} from 'assemble-react-bulma'
6
7  function App () {
8    return (
9      <div className='container is-fluid'>
10       <Message definition="is-dark" header="UFSC - CTC - INE - INE5646 :: App Compra Vende :: Vendedor">
11         <Columns className='columns'>
12           <Column className='column'>
13             <VerificaPedidos/>
14           </Column>
15         </Columns>
16       </Message>
17     </div>
18   )
19 }
20
21 export default App
```

**Fonte**: O próprio autor,2020

Na Figura 133, pode-se observar que existem 9 linhas referente ao código HTML (9-17), 9 tags, além do código original possuir um total de 21 linhas.

O componente App (App Venda) apresentou uma redução de 30.77% no número de tags, redução de 35.71% no número de linhas HTML e 16.00% no número total de linhas do componente.

### *5.1.3.8 ProcessaPedido*

O componente ProcessaPedido utilizou os componentes Card e Button da presente solução para o *refactoring*.

**Figura 134** - Imagem original componente ProcessaPedido


**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 135** - Imagem original após refactoring componente ProcessaPedido


**Fonte**: O próprio autor,2020

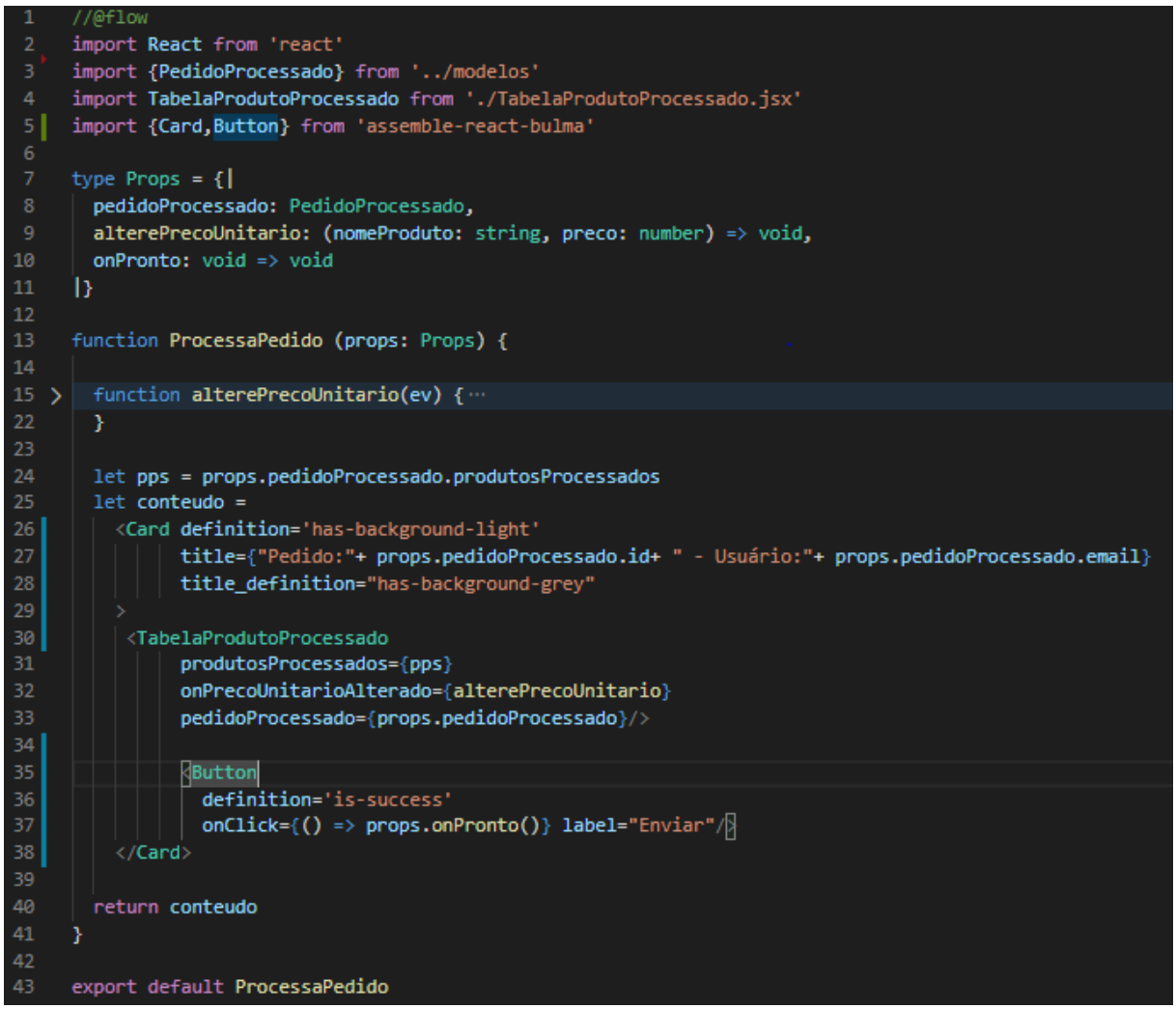
**Figura 136** - Código original componente ProcessaPedido

```
1   //@FIXME
2   import React from 'react'
3   import {PedidoProcessado} from '../modelos'
4   import TabelaProdutoProcessado from './TabelaProdutoProcessado.jsx'
5
6   type Props = {|
7     pedidoProcessado: PedidoProcessado,
8     alterePrecoUnitario: (nomeProduto: string, preco: number) => void,
9     onPronto: void => void
10  |}
11
12  function ProcessaPedido (props: Props) {
13
14    function alterePrecoUnitario(ev) {···
21    }
22
23    let pps = props.pedidoProcessado.produtosProcessados
24    let conteudo =
25      <div className='card has-background-light'>
26        <div className='card-header'>
27          <h3 className='card-header-title has-background-grey'>
28            Pedido: {props.pedidoProcessado.id} - Usuário: {props.pedidoProcessado.email}
29          </h3>
30        </div>
31        <div className='card-content'>
32          <TabelaProdutoProcessado
33            produtosProcessados={pps}
34            onPrecoUnitarioAlterado={alterePrecoUnitario}
35            pedidoProcessado={props.pedidoProcessado}/>
36          <div>
37            <button
38              className='button is-success'
39              onClick={() => props.onPronto()}>Enviar</button>
40          </div>
41        </div>
42      </div>
43
44    return conteudo
45  }
46
47  export default ProcessaPedido
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na Figura 136, pode-se observar que existem 19 linhas referente ao código HTML (25-44), 13 tags, além do código original possuir um total de 47 linhas.

**Figura 137** - Código original refatorado ProcessaPedido



```
1  //@flow
2  import React from 'react'
3  import {PedidoProcessado} from '../modelos'
4  import TabelaProdutoProcessado from './TabelaProdutoProcessado.jsx'
5  import {Card,Button} from 'assemble-react-bulma'
6
7  type Props = {|
8    pedidoProcessado: PedidoProcessado,
9    alterePrecoUnitario: (nomeProduto: string, preco: number) => void,
10   onPronto: void => void
11 |}
12
13 function ProcessaPedido (props: Props) {
14
15   function alterePrecoUnitario(ev) {···
22   }
23
24   let pps = props.pedidoProcessado.produtosProcessados
25   let conteudo =
26     <Card definition='has-background-light'
27           title={"Pedido:"+ props.pedidoProcessado.id+ " - Usuário:"+ props.pedidoProcessado.email}
28           title_definition="has-background-grey"
29     >
30      <TabelaProdutoProcessado
31           produtosProcessados={pps}
32           onPrecoUnitarioAlterado={alterePrecoUnitario}
33           pedidoProcessado={props.pedidoProcessado}/>
34
35           <Button
36             definition='is-success'
37             onClick={() => props.onPronto()} label="Enviar"/>
38     </Card>
39
40   return conteudo
41 }
42
43 export default ProcessaPedido
```

**Fonte**: O próprio autor,2020

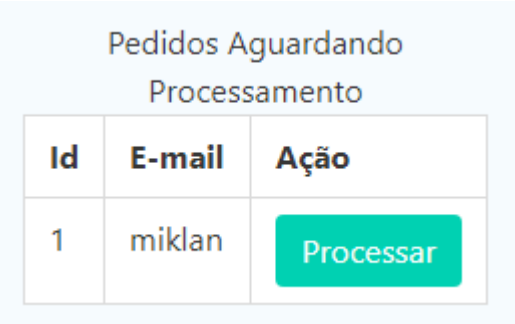
Na Figura 137, pode-se observar que existem 15 linhas referente ao código HTML (26-40), 5 tags, além do código refatorado possuir um total de 43 linhas.

O componente ProcessaPedido apresentou uma redução de 69.23% no número de tags, redução de 21.05% no número de linhas HTML e 8.51% no número total de linhas do componente.

### *5.1.3.9 TabelaPedidos*

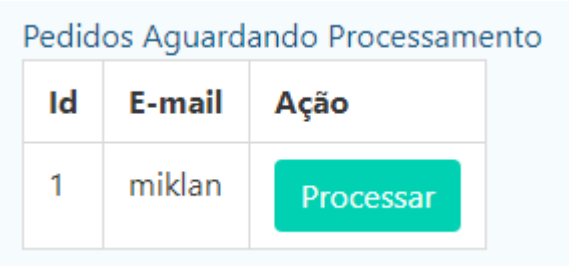
No refactoring de TabelaPedidos foram utilizados os componentes Button e TableQuery.

**Figura 138** - Imagem original componente TabelaPedidos




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 139** - Imagem após refactoring componente TabelaPedidos




**Fonte**: O próprio autor, 2020

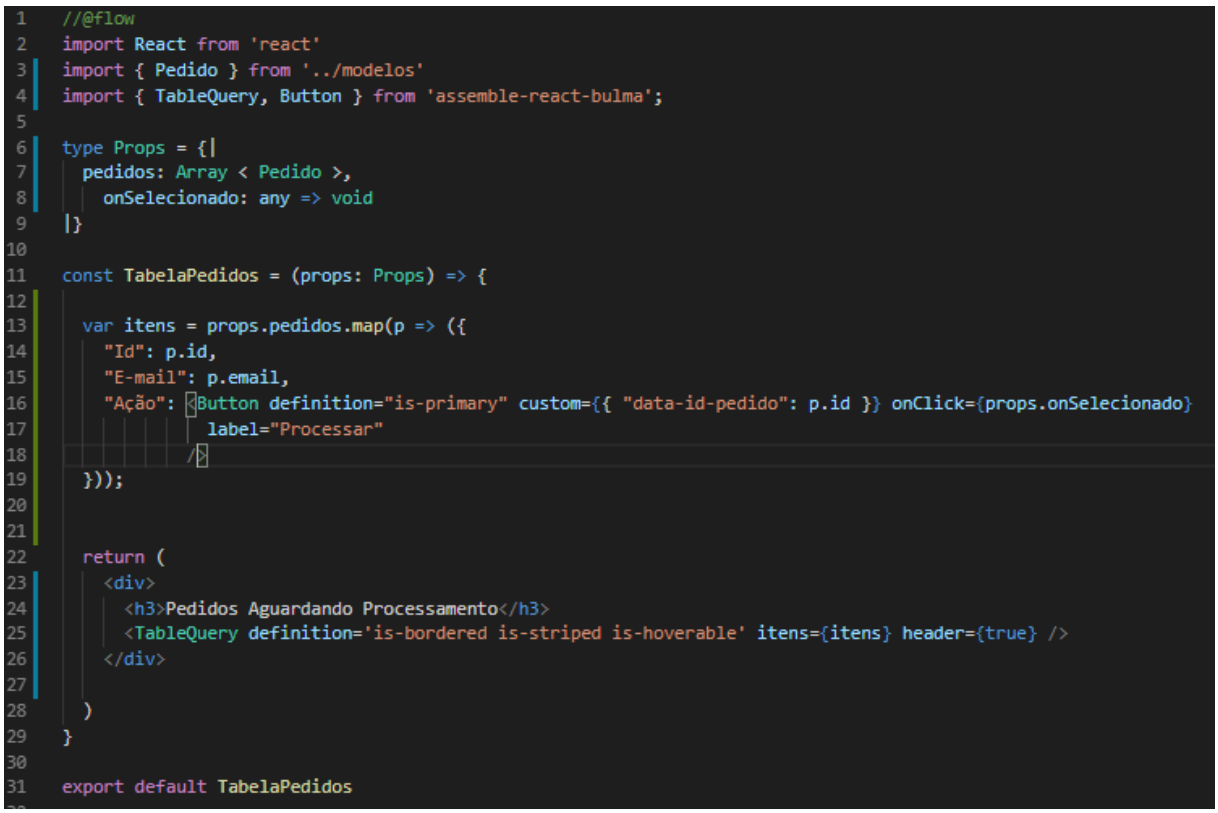
**Figura 140** - Código original TabelaPedidos

```
1  //@flow
2  import React from 'react'
3  import {Pedido} from '../modelos'
4
5  type Props = {|
6    pedidos: Array<Pedido>,
7    onSelecionado: any => void
8  |}
9
10 const TabelaPedidos = (props: Props) => {
11   return (
12     <table className='table is-bordered is-striped is-hoverable'>
13       <caption>Pedidos Aguardando Processamento</caption>
14       <thead>
15         <tr><th>Id</th><th>E-mail</th><th>Ação</th></tr>
16       </thead>
17       <tbody>
18         {props.pedidos.map(p =>
19           <tr key={p.id}>
20             <td>{p.id}</td>
21             <td>{p.email}</td>
22             <td><button className='button is-primary'
23               data-id-pedido={p.id}
24               onClick={props.onSelecionado}>
25                   Processar
26             </button>
27             </td>
28           </tr>
29         )}
30       </tbody>
31     </table>
32   )
33 }
34
35 export default TabelaPedidos
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na **Figura *140***, pode-se observar que existem 19 linhas referente ao código HTML (12-31), 26 tags, além do código original possuir um total de 35 linhas

**Figura 141** - Código refatorado TabelaPedidos



```
1  //@flow
2  import React from 'react'
3  import { Pedido } from '../modelos'
4  import { TableQuery, Button } from 'assemble-react-bulma';
5
6  type Props = {|
7    pedidos: Array < Pedido >,
8      onSelecionado: any => void
9  |}
10
11 const TabelaPedidos = (props: Props) => {
12
13   var itens = props.pedidos.map(p => ({
14     "Id": p.id,
15     "E-mail": p.email,
16     "Ação": <Button definition="is-primary" custom={{ "data-id-pedido": p.id }} onClick={props.onSelecionado}
17               label="Processar"
18             />
19   }));
20
21
22   return (
23     <div>
24       <h3>Pedidos Aguardando Processamento</h3>
25       <TableQuery definition='is-bordered is-striped is-hoverable' itens={itens} header={true} />
26     </div>
27
28   )
29 }
30
31 export default TabelaPedidos
```

**Fonte**: O próprio autor, 2020

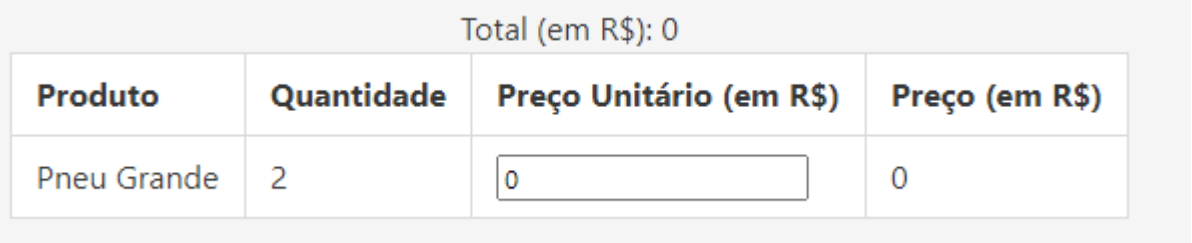
Na **Figura *141*** e Figura *137*, pode-se observar que existem 7 linhas referente ao código HTML (16-18 e 22-25), 7 tags, além do código refatorado possuir um total de 30 linhas.

O componente TabelaPedidos apresentou uma redução de 76.92% no número de tags, redução de 63.16% no número de linhas HTML e 14.29% no número total de linhas do componente.

### *5.1.3.10 TabelaProdutoProcessado*

No refactoring de TabelaProdutosProcessados foram utilizados os componentes TableQuery e Input.

**Figura 142** - Imagem original TabelaProdutoProcessado




**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 143** - Imagem após refactoring TabelaProdutoProcessado


**Fonte**: O próprio autor,2020

**Figura 144** - Código original TabelaProdutoProcessado

```
import React from 'react'
import {ProdutoProcessado, PedidoProcessado} from '../modelos'

type Props = {
  produtosProcessados: Array<ProdutoProcessado>,
  pedidoProcessado: PedidoProcessado,
  onPrecoUnitarioAlterado: any => void
}

const TabelaProdutoProcessado = (props: Props) => {
  return (
    <table className='table is-bordered is-striped is-hoverable'>
      <thead>
        <tr>
          <th>Produto</th>
          <th>Quantidade</th>
          <th>Preço Unitário (em R$)</th>
          <th>Preço (em R$)</th>
        </tr>
      </thead>
      <tbody>
        {props.produtosProcessados.map(pp =>
          <tr key={pp.produto.nome}>
            <td>{pp.produto.nome}</td>
            <td>{pp.produto.quantidade}</td>
            <td>
              <input type='number' min='0' step='0.01'
                value={pp.precoUnitario}
                onChange={props.onPrecoUnitarioAlterado}
                name={pp.produto.nome}/>
            </td>
            <td>{pp.custo()}</td>
          </tr>
        )}
      </tbody>
      <caption>Total (em R$): {props.pedidoProcessado.custo()}</caption>
    </table>
  )
}
export default TabelaProdutoProcessado
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na **Figura *144***, pode-se observar que existem 26 linhas referente ao código HTML (13-38), 29 tags, além do código original possuir um total de 41 linhas

**Figura 145** - Código refatorado TabelaProdutoProcessado

```
1  //@flow
2  import React from 'react'
3  import { ProdutoProcessado, PedidoProcessado } from '../modelos'
4  import { TableQuery, Input } from 'assemble-react-bulma';
5
6  type Props = {|
7    produtosProcessados: Array < ProdutoProcessado >,
8      pedidoProcessado: PedidoProcessado,
9        onPrecoUnitarioAlterado: any => void
10 |}
11
12 const TabelaProdutoProcessado = (props: Props) => {
13   var itens = props.produtosProcessados.map(pp => ({
14     "Produto": pp.produto.nome,
15     "Quantidade": pp.produto.quantidade,
16     "Preço Unitário (em R$)": <input type='number' min='0' step='0.01'
17                                 value={pp.precoUnitario} onChange={props.onPrecoUnitarioAlterado}
18                                 name={pp.produto.nome} />,
19     "Preço (em R$)": pp.custo()
20   }))
21
22   return (
23     <>
24       <h3>Total (em R$): {props.pedidoProcessado.custo()}</h3>
25       <TableQuery itens={itens} definition="is-bordered is-striped is-hoverable" header={true} />
26     </>
27   )
28 }
29 export default TabelaProdutoProcessado
```

**Fonte**: O próprio autor,2020

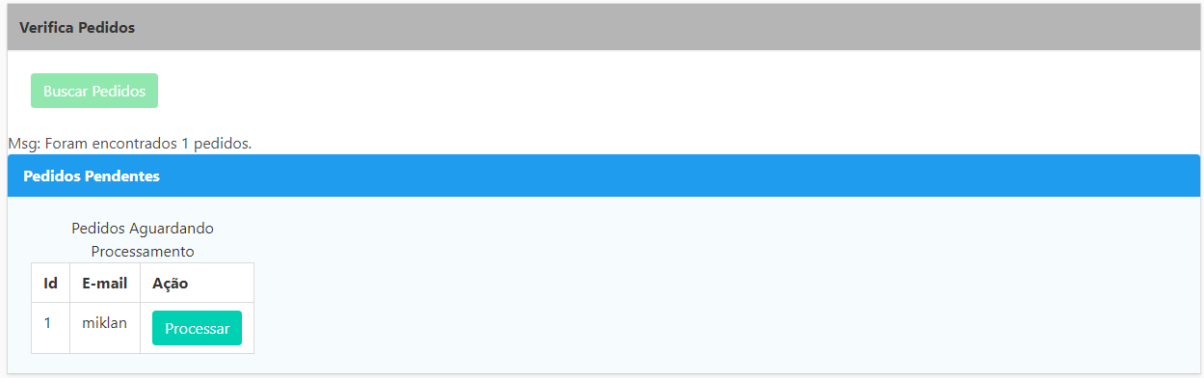
Na **Figura 145** pode-se observar que existem 7 linhas referente ao código HTML (16-18 e 23-26), 6 tags, além do código refatorado possuir um total de 29 linhas

O componente TabelaProdutosProcessados apresentou uma redução de 79.31% no número de tags, redução de 73.08% no número de linhas HTML e 29.27% no número total de linhas do componente.

### *5.1.3.11 VerificaPedidos*

O componente VerificaPedidos utilizou os componentes Message e Card da presente solução

**Figura 146** - Imagem original VerificaPedidos



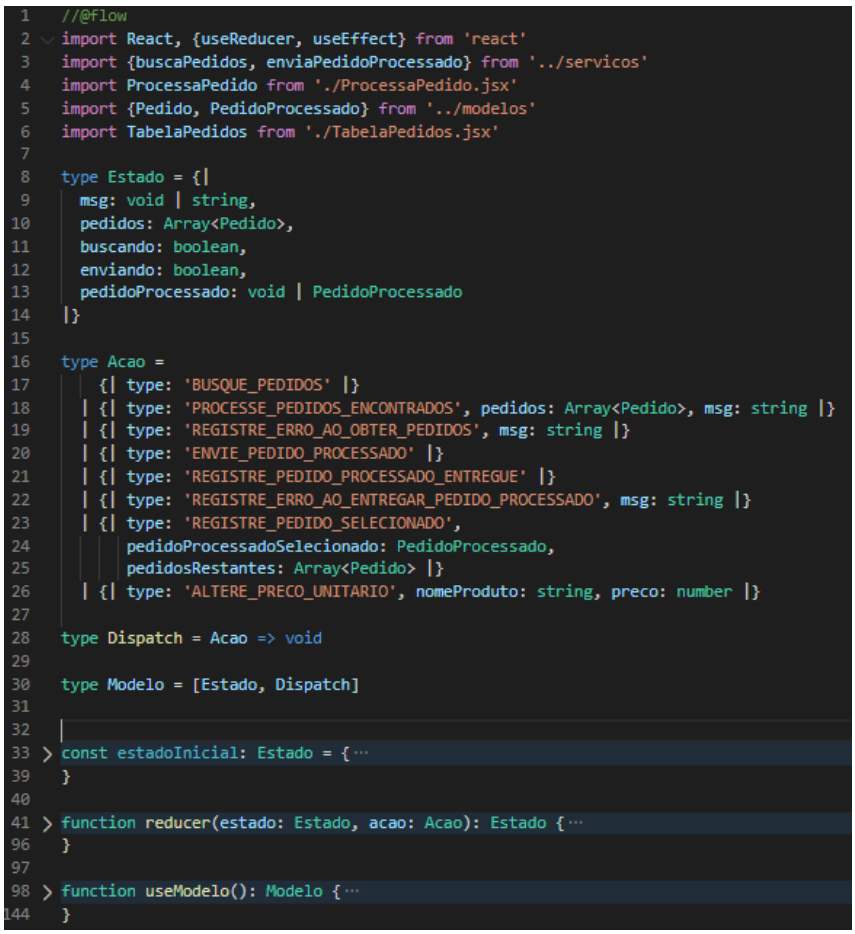

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 147** - Imagem após refactoring VerificaPedidos


**Fonte**: O próprio autor,2020

O resultado gráfico após o refactoring é igual ao resultado original do componente.

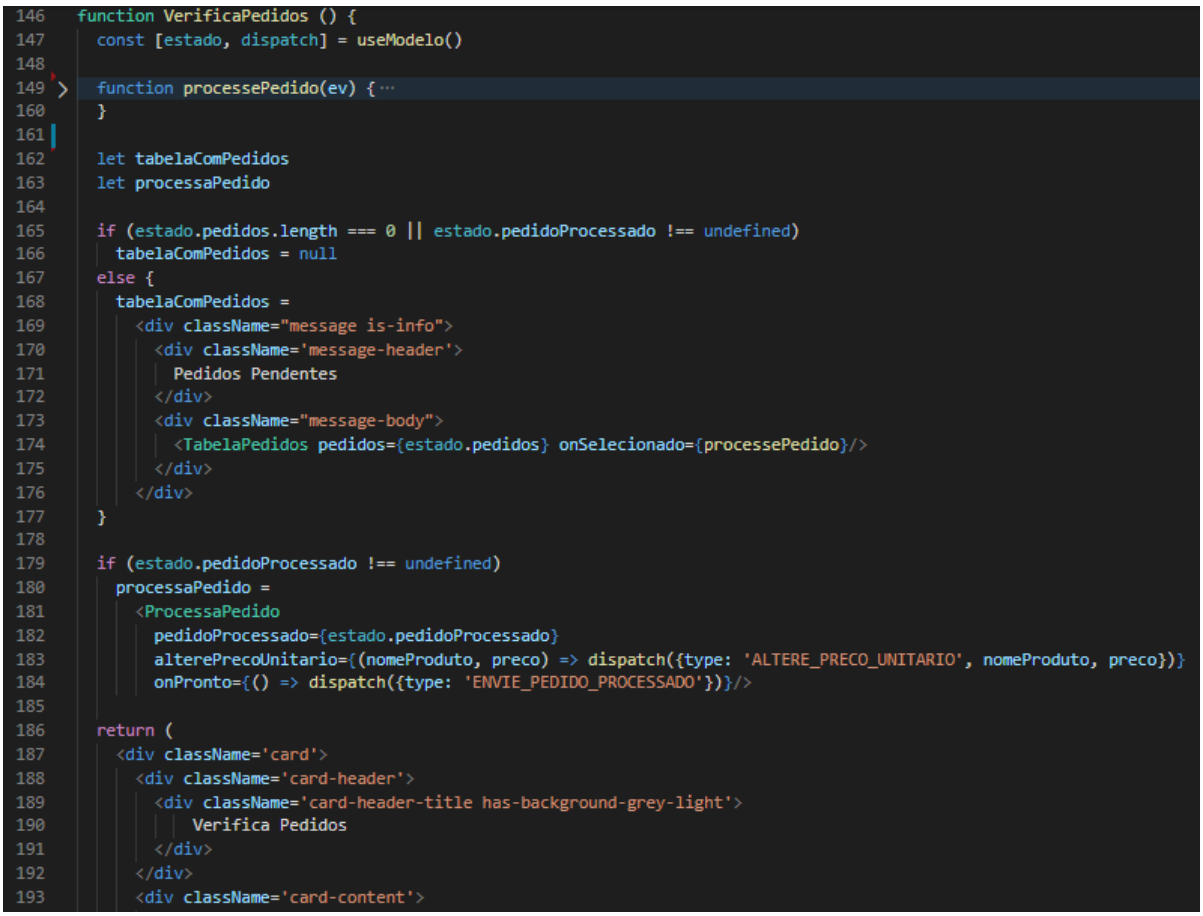
**Figura 148** - Código original VerificaPedidos I



```
1   //@flow
2   import React, {useReducer, useEffect} from 'react'
3   import {buscaPedidos, enviaPedidoProcessado} from '../servicos'
4   import ProcessaPedido from './ProcessaPedido.jsx'
5   import {Pedido, PedidoProcessado} from '../modelos'
6   import TabelaPedidos from './TabelaPedidos.jsx'
7
8   type Estado = {|
9     msg: void | string,
10    pedidos: Array<Pedido>,
11    buscando: boolean,
12    enviando: boolean,
13    pedidoProcessado: void | PedidoProcessado
14  |}
15
16  type Acao =
17      {| type: 'BUSQUE_PEDIDOS' |}
18    | {| type: 'PROCESSE_PEDIDOS_ENCONTRADOS', pedidos: Array<Pedido>, msg: string |}
19    | {| type: 'REGISTRE_ERRO_AO_OBTER_PEDIDOS', msg: string |}
20    | {| type: 'ENVIE_PEDIDO_PROCESSADO' |}
21    | {| type: 'REGISTRE_PEDIDO_PROCESSADO_ENTREGUE' |}
22    | {| type: 'REGISTRE_ERRO_AO_ENTREGAR_PEDIDO_PROCESSADO', msg: string |}
23    | {| type: 'REGISTRE_PEDIDO_SELECIONADO',
24          pedidoProcessadoSelecionado: PedidoProcessado,
25          pedidosRestantes: Array<Pedido> |}
26    | {| type: 'ALTERE_PRECO_UNITARIO', nomeProduto: string, preco: number |}
27
28  type Dispatch = Acao => void
29
30  type Modelo = [Estado, Dispatch]
31
32  |
33  > const estadoInicial: Estado = {···
39  }
40
41  > function reducer(estado: Estado, acao: Acao): Estado {···
96  }
97
98  > function useModelo(): Modelo {···
144 }
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

**Figura 149** - Código original VerificaPedidos II



```
function VerificaPedidos () {
  const [estado, dispatch] = useModelo()

  function processePedido(ev) {···
  }

  let tabelaComPedidos
  let processaPedido

  if (estado.pedidos.length === 0 || estado.pedidoProcessado !== undefined)
    tabelaComPedidos = null
  else {
    tabelaComPedidos =
      <div className="message is-info">
        <div className='message-header'>
          Pedidos Pendentes
        </div>
        <div className="message-body">
          <TabelaPedidos pedidos={estado.pedidos} onSelecionado={processePedido}/>
        </div>
      </div>
  }

  if (estado.pedidoProcessado !== undefined)
    processaPedido =
      <ProcessaPedido
        pedidoProcessado={estado.pedidoProcessado}
        alterePrecoUnitario={(nomeProduto, preco) => dispatch({type: 'ALTERE_PRECO_UNITARIO', nomeProduto, preco})}
        onPronto={() => dispatch({type: 'ENVIE_PEDIDO_PROCESSADO'})}/>

  return (
    <div className='card'>
      <div className='card-header'>
        <div className='card-header-title has-background-grey-light'>
          Verifica Pedidos
        </div>
      </div>
      <div className='card-content'>
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

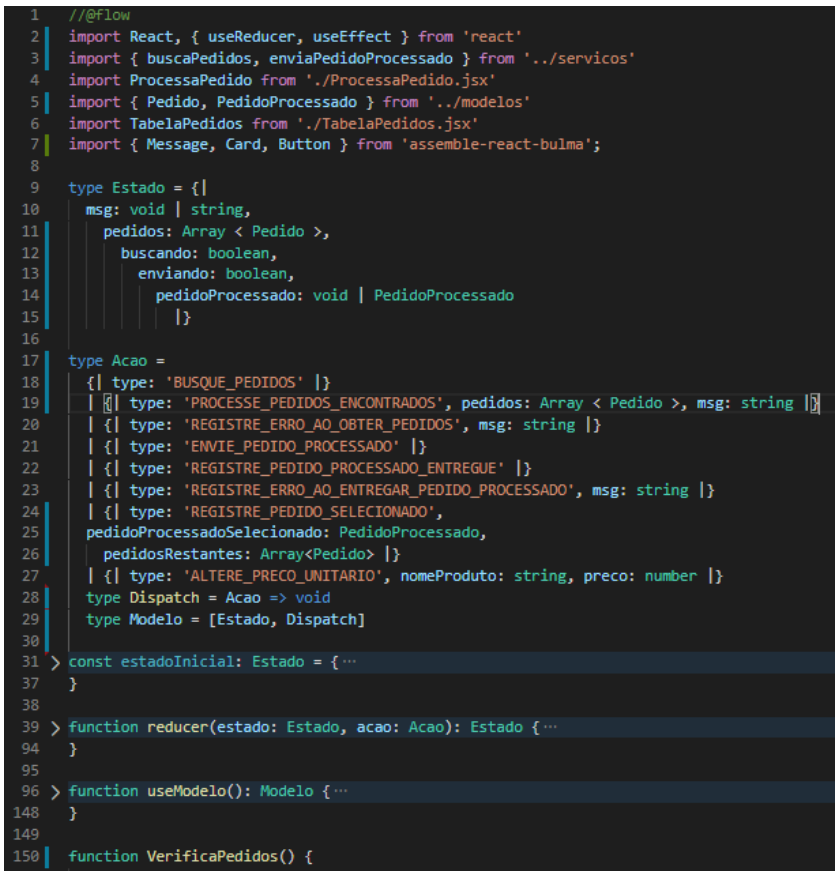
**Figura 150** - Código original VerificaPedidos III

```
194                <button
195                  className='button is-success'
196                  onClick={() => dispatch({type: 'BUSQUE_PEDIDOS'})}
197                  disabled={
198                    estado.buscando ||
199                    estado.pedidos.length > 0 ||
200                    estado.pedidoProcessado !== undefined}>
201                    Buscar Pedidos
202                </button>
203              </div>
204              <div>
205                <h4>
206                    Msg: {estado.msg}
207                </h4>
208              </div>
209              <div>
210                {tabelaComPedidos}
211              </div>
212              <div>
213                {processaPedido}
214              </div>
215
216            </div>
217          )
218
219        }
220
221        export default VerificaPedidos
```

**Fonte**: https://github.com/ine5646-tarefas/ine5646-tarefas-20192/tree/master/10-app_compra_vende

Na **Figura *149*** e **Figura *150***, pode-se observar que existem 42 linhas referente ao código HTML (169-176,181-184 e 187-216), 26 tags, além do código original possuir um total de 221 linhas.

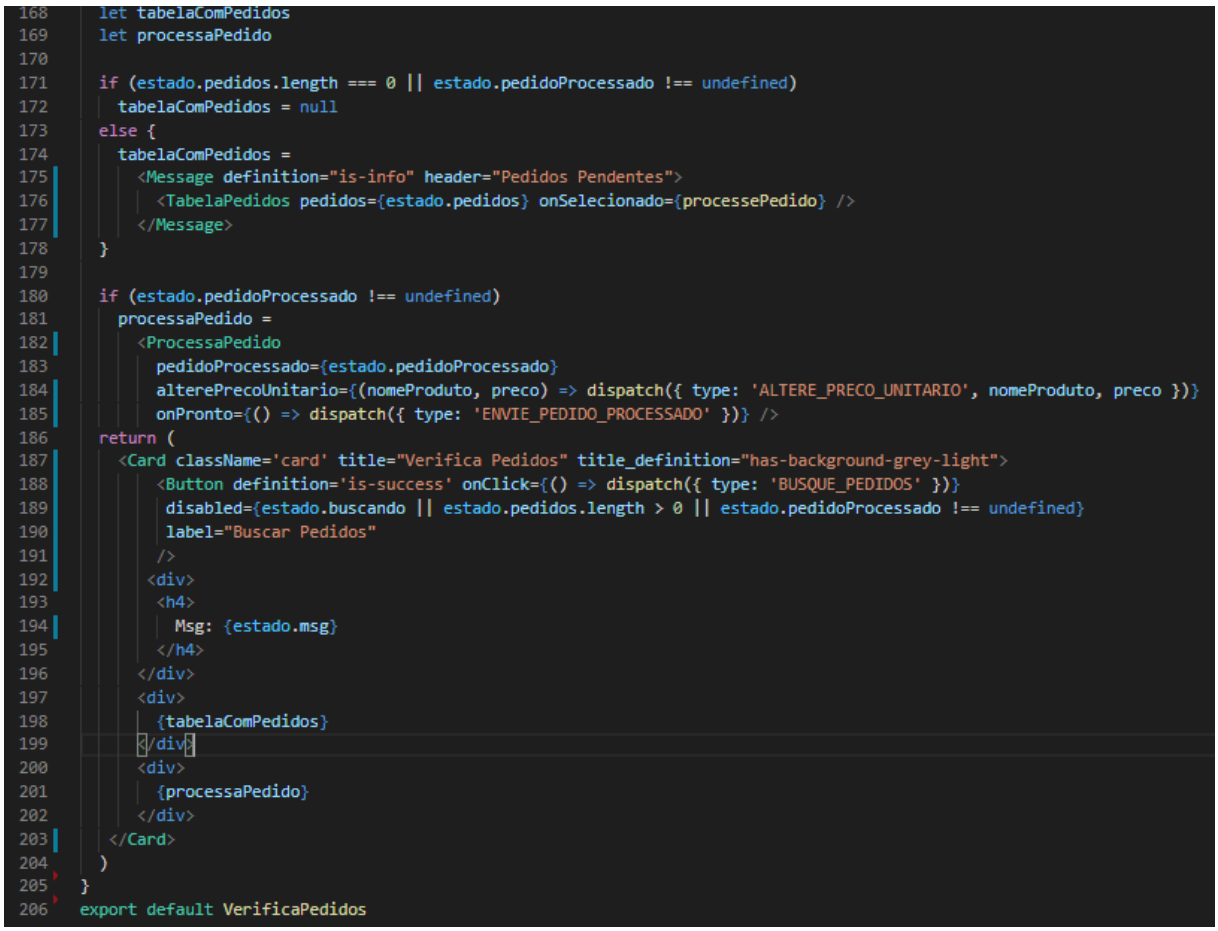
**Figura 151** - Código refatorado VerificaPedidos I



```
//@flow
import React, { useReducer, useEffect } from 'react'
import { buscaPedidos, enviaPedidoProcessado } from '../servicos'
import ProcessaPedido from './ProcessaPedido.jsx'
import { Pedido, PedidoProcessado } from '../modelos'
import TabelaPedidos from './TabelaPedidos.jsx'
import { Message, Card, Button } from 'assemble-react-bulma';

type Estado = {|
  msg: void | string,
    pedidos: Array < Pedido >,
      buscando: boolean,
        enviando: boolean,
          pedidoProcessado: void | PedidoProcessado
            |}

type Acao =
  {| type: 'BUSQUE_PEDIDOS' |}
  | {| type: 'PROCESSE_PEDIDOS_ENCONTRADOS', pedidos: Array < Pedido >, msg: string |}
  | {| type: 'REGISTRE_ERRO_AO_OBTER_PEDIDOS', msg: string |}
  | {| type: 'ENVIE_PEDIDO_PROCESSADO' |}
  | {| type: 'REGISTRE_PEDIDO_PROCESSADO_ENTREGUE' |}
  | {| type: 'REGISTRE_ERRO_AO_ENTREGAR_PEDIDO_PROCESSADO', msg: string |}
  | {| type: 'REGISTRE_PEDIDO_SELECIONADO',
  pedidoProcessadoSelecionado: PedidoProcessado,
    pedidosRestantes: Array<Pedido> |}
  | {| type: 'ALTERE_PRECO_UNITARIO', nomeProduto: string, preco: number |}
  type Dispatch = Acao => void
  type Modelo = [Estado, Dispatch]

> const estadoInicial: Estado = {···
  }

> function reducer(estado: Estado, acao: Acao): Estado {···
  }

> function useModelo(): Modelo {···
  }

  function VerificaPedidos() {
```

**Fonte**: O próprio autor,2020

**Figura 152** - Código refatorado VerificaPedidos II



```
168   let tabelaComPedidos
169   let processaPedido
170
171   if (estado.pedidos.length === 0 || estado.pedidoProcessado !== undefined)
172     tabelaComPedidos = null
173   else {
174     tabelaComPedidos =
175       <Message definition="is-info" header="Pedidos Pendentes">
176         <TabelaPedidos pedidos={estado.pedidos} onSelecionado={processePedido} />
177       </Message>
178   }
179
180   if (estado.pedidoProcessado !== undefined)
181     processaPedido =
182       <ProcessaPedido
183         pedidoProcessado={estado.pedidoProcessado}
184         alterePrecoUnitario={(nomeProduto, preco) => dispatch({ type: 'ALTERE_PRECO_UNITARIO', nomeProduto, preco })}
185         onPronto={() => dispatch({ type: 'ENVIE_PEDIDO_PROCESSADO' })} />
186   return (
187     <Card className='card' title="Verifica Pedidos" title_definition="has-background-grey-light">
188         <Button definition='is-success' onClick={() => dispatch({ type: 'BUSQUE_PEDIDOS' })}
189          disabled={estado.buscando || estado.pedidos.length > 0 || estado.pedidoProcessado !== undefined}
190          label="Buscar Pedidos"
191         />
192        <div>
193         <h4>
194           Msg: {estado.msg}
195         </h4>
196       </div>
197       <div>
198         {tabelaComPedidos}
199       </div>
200       <div>
201         {processaPedido}
202       </div>
203    </Card>
204   )
205 }
206 export default VerificaPedidos
```

**Fonte**: O próprio autor,2020

Na **Figura *152***, pode-se observar que existem 25 linhas referente ao código HTML (175-177,182-185 e 187-203), 16 tags, além do código original possuir um total de 206 linhas.

O componente VerificaPedidos apresentou uma redução de 42.31% no número de tags, redução de 16.67% no número de linhas HTML e 6.79% no número total de linhas do componente.

### *5.1.3.12 Métricas Refactoring*

No presente subtópico são apresentadas as métricas do refactoring aplicado nos componentes de App Compra e Vende.

**Tabela 12** - Legenda de App Compra e Vende

| Sigla | Descrição |
|---|---|
| L O | Total de linhas do código original |
| L R | Total de linhas do código refatorado |
| TO | Total de tags do código original |
| TR | Total de tags do código refatorado |
| LHO | Total de linhas HTML do código original |
| LHR | Total de linhas HTML do código refatorado |
| %LH | Porcentagem de redução do código HTML |
| %LT | Porcentagem de redução do código total |
| %T | Porcentagem de redução do número de tags |

**Fonte**: O próprio autor, 2020

**Tabela 13** - Métricas refactoring de App Compra e Vende

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| App | 24 | 20 | 12 | 8 | 13 | 8 | 16.67 | 33.33 | 38.46 |
| FazPedido | 248 | 210 | 54 | 16 | 114 | 55 | 15.32 | 70.37 | 51.75 |
| TabelaProdutos | 38 | 27 | 24 | 2 | 20 | 4 | 28.95 | 91.67 | 80.00 |
| TabelaProdutosProcessados | 34 | 25 | 26 | 1 | 19 | 1 | 26.47 | 96.15 | 94.74 |
| MostraPedidosProcessados | 66 | 52 | 32 | 10 | 32 | 10 | 21.21 | 68.75 | 68.75 |
| VerificaPedido | 136 | 115 | 36 | 13 | 41 | 20 | 15.44 | 63.89 | 51.22 |
| App(App Venda) | 25 | 21 | 13 | 9 | 14 | 9 | 16.00 | 30.77 | 35.71 |
| Processa Pedido | 47 | 43 | 13 | 4 | 19 | 15 | 8.51 | 69.23 | 21.05 |
| TabelaPedidos | 35 | 30 | 26 | 6 | 19 | 7 | 14.29 | 76.92 | 63.16 |
| TabelaProdutoProcessado | 41 | 29 | 29 | 6 | 26 | 7 | 29.27 | 79.31 | 73.08 |
| VeificaPedidos | 221 | 206 | 26 | 15 | 42 | 35 | 6.79 | 42.31 | 16.67 |

**Fonte**: O próprio autor, 2020

### 5.1.4 Considerações Finais

Na presente seção, um comparativo das aplicações refatoradas será realizado, destacando resultados interessantes, características observadas e a conclusão dos resultados obtidos.

No *refactoring* do App Lembretes destacou-se a redução de 27.27% das tags do componente MostraLembretes, a redução de 64,44% do componente Login e a redução de 11,59% das linhas totais do componente Login. A redução de 27,27% do

componente MostraLembretes é a menor de todo os resultados obtidos, isso pelo fato do componente original ser muito simples e existir basicamente uma equivalência entre original e refatorado, mesmo assim, a redução foi maior do que 25% conforme definido no objetivo secundário do presente trabalho. A redução de 64,44% do componente Login mostra um bom resultado obtido com a solução e a redução de 11,59% de linhas totais mostra uma boa redução do código total apenas utilizando a solução.

O *refactoring* do App Vulcões demonstra uma boa redução de tags, apresentando 68,29% de redução.

O *refactoring* do App Compra e Vende apresenta os melhores resultados, com 96,15% de redução de tags no componente TabelaProdutosProcessad**os** e diminuição de 29,27% na quantidade de linhas totais do componente TabelaProdutoProcessad**o,** além de diversos outros bons resultados.

Se faz necessário pontuar duas características observadas nos resultados obtidos: A relação entre complexidade do componente com a diminuição do código e a métrica “diminuição de linhas totais” que em todos os resultados se manteve maior igual a 0. Ao longo das aplicações da solução, notou-se que os componentes com mais código, com um front-end mais complexo, apresentaram diminuição de código superior se comparado com componentes de complexidades inferiores. A métrica de linhas totais apresentou uma característica interessante, em todos os resultados observados, não existiu aumentou do código total, ou seja, não foi necessário programar mais para diminuir o código referente a interface gráfica do componente.

Com a apresentação dos resultados exibidos no presente capítulo, evidencia-se o cumprimento dos objetivos gerais e específico de fornecer uma biblioteca de componentes React que diminui em aproximadamente 25% das tags HTML em componentes refatorados.

# 6 CONCLUSÃO

O presente trabalho buscou uma forma de diminuir a quantidade de código necessário ao utilizar a biblioteca React e o framework Bulma para se desenvolver aplicações Web, através de uma biblioteca de componentes *open source* criada especificamente para a integração React/Bulma.

Com o desenvolvimento do trabalho foi possível concluir que a solução diminui de maneira considerável a quantidade de código HTML necessário para construção de interfaces gráficas utilizando React e Bulma, permitindo criar soluções completas com menor digitação de código, atingindo também o objetivo de reduzir aproximadamente 25% das tags após *refactoring*.

Apesar da redução de código, alguns pontos ao decorrer do desenvolvimento foram considerados limitadores e negativos da presente solução.

O principal ponto limitador na solução desenvolvida é a dificuldade de prever quais componentes poderiam ser úteis ou desejados pelos usuários da solução, uma dificuldade de expansão da biblioteca de componentes a partir da possível necessidade dos usuários, que podem ser dos mais variados tipos.

A curva de aprendizado do React e as tecnologias envolvidas no seu uso podem ser fatores negativos ao aprendizado da presente solução. Essa conclusão está suportada pela experiência do autor ao utilizar as tecnologias diariamente sem possuir experiência prévia. No desenvolvimento da presente solução, diversos foram os problemas no aprendizado de tecnologias e conceitos, mas de todas as negativas no aprendizado, o React se torna o problema mais complexo, por funcionar através de um paradigma não tão comum (programação reativa), a tecnologia mostra uma curva de aprendizado grande e conceitos implícitos que só foram entendidos mediante a grande experiência prática. Observando as dificuldades antes citadas, é bem provável que usuários que não possuam conhecimentos básicos do Node JS, React, *Hooks* e gerência de estado passem por alguns problemas no uso da presente solução.

Durante o desenvolvimento, alguns trabalhos futuros foram pensados como possíveis melhorias para a presente solução. Os dois considerados mais interessantes pelo autor são a criação de uma biblioteca focada na parte comportamental do React/Bulma e a criação de uma biblioteca de componentes

dependentes mais complexos a partir dos componentes implementados no presente trabalho.

A presente solução por ser focada especificamente na redução do código HTML, acaba por não entregar uma grande redução do código geral de um componente, isto é, da parte comportamental de um componente. Observando essa limitação, como trabalho futuro sugere-se uma expansão do presente trabalho que lide com o código que não seja HTML de um componente React/Bulma, como por exemplo a utilização dos *hooks* e eventos do componente.

Outra sugestão de trabalho futuro é a elaboração de componentes React/Bulma focados somente em componentes dependentes de complexidade elevada. Conforme observado no *refactoring* apresentado na seção 5.1 os componentes que são mais complexos, que abstraem uma grande quantidade de componentes, são os que mais apresentam reuso de software, logo, recomenda-se um trabalho que o objetivo seja a elaboração de componentes complexos de diversos tipos e domínios de uso que utilizem como base os implementados no presente trabalho.

O resultado final do presente trabalho conta com a biblioteca pronta para o uso e disponível para download no repositório **NPM**, onde para encontrar o pacote deve-se informar as palavras chaves “assemble-react-bulma”. Pontua-se que é uma biblioteca open source e oferecida com a licença **MIT**.

# REFERÊNCIAS


ALVARO, Alexandre *et al*. **C.R.U.I.S.E**: component reuse in software engineering. Recife: [s.n.], 2007. 210 p. Disponível em:< https://www.researchgate.net/publication/236660495_CRUISE_Component_Reuse_in_Software_Engineering>

BANKS, Alex; PORCELLO, Eve. **Learning React**: functional web development with react and redux. Sebastopol: O'reilly Media, 2017. 350 p.

BULMA**. Bulma:** free, open source, and modern CSS framework based on Flexbox.2020. Disponível em :<https://bulma.io/>. Acesso em: 30/01/2020.

FAYAD, Mohamed; FLOOD, Charles. Software Reuse Knowledge Map. **Ijars: International Journal of Engineering.** San Jose, p. 1-26. set. 2016. Disponível em:<https://www.researchgate.net/publication/309271508_Software_Reuse_Knowledge_Map>. Acesso em: 24/01/2020

FAYAD, Mohamed; SCHMIDT, Douglas. Object-Oriented Application Frameworks. **Communications Of The Acm.** [s. L.], p. 32-38. out. 1997.Disponível em:<https://www.researchgate.net/publication/215617675_ObjectOriented_Application_Frameworks>

FLANAGAN, David. **JavaScript**: the definitive guide. 6. ed. Sebastopol: O'reilly Media, 2011. 1078 p.

FLOW. **Flow**: A Static Type Checker For JavaScript.2020. Disponível em:< https://flow.org/> Acessado em: 26/06/2020.

GITHUB. **The State of the Octoverse**.2019. Disponível em: <https://octoverse.github.com/>. Acesso em: 20 out. de 2019.

JEST. **Jest**: Delightful Javascript Testing. 2020. Disponível em: <https://jestjs.io/>. Acessado em: 16/11/2020

JOHNSON, Ralph; FOOTE, Brian. Designing Reusable Classes. **Joop: Journal of Object-Oriented Programming.** Illinois, p. 22-35. jun. 1988.Disponível em:<https://www.researchgate.net/publication/215446177_Designing_Reusable_Classes> Acessado em: 24/01/2020.

KEITH, Jeremy; SAMBELLS, Jeffrey**. DOM Scripting**: Web design with javascript and the document object model. 2. ed. Nova York: Apress, 2010. 319 p.

MUSCIANO, Chuck; KENNEDY, Bill. **HTML & XHTML**: The definitive guide. 4. ed. [s. L.]: O'Reilly Media, 2000. 677 p.

MAYE'R, Eric. **Cascading Style Sheets**: the definitive guide. 2. ed. Sebastopol: O'Reilly Media.2004.528 p.

PRESSMAN, Roger S. **Engenharia de Software:** Uma Abordagem Profissional. 7º edição. Porto Alegre: AMGH Editora LTDA.2011.180 p.

REACT**. React:** A JavaScript library for building user interfaces. 2020. Disponível em: <https://pt-br.reactjs.org/>. Acessado em: 30/01/2020

REDUX. **Redux:** A Predictable State Container for JS Apps**.** 2020. Disponível em: <https://redux.js.org/>. Acessado em: 27/03/2020

W3. **HTML**: The Language for building web pages.2020. Disponível em:< https://www.w3schools.com/>. Acessado em: 24/01/2020.

W3. **Document Object Model Specification**.1998. Disponível em: < https://www.w3.org/TR/1998/WD-DOM-19980416/>

SAMETINGER, Johannes**. Software Engineering with Reusable Components**. Linz. [s.n].1997.272 p.

SAMY, Maurício S. **JavaScript - Guia do Programador**. São Paulo. Novatec Editora.2010.608 p.

SAMY, Maurício S**. CSS3:** Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec Editora. 2012.496 p.

SASS. **SASS:** Syntactically Awesome Style Sheets**.**2020**.** Disponível em:< https://sass-lang.com/> Acesso em : 30 de jun. de 2020

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 p. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

SOMMERVILLE, Ian. **Engenharia de Software**. 9º edição. São Paulo: Pearson Prentice Hall.2011.529 p.

STACKOVERFLOW. **Most Popular Technologies**,2020. Disponível em: < https://insights.stackoverflow.com/survey/2020#most-popular-technologies > Acesso em: 16 de nov. de 2020.

STEFANOV, Stoyan**. React Up & Running:** Building Web Applications. 1º edição. United States: O'Reilly Media.2016.201 p.

WADLER, Philip; BIRD, Richard. **Introduction to functional programming**. 1º edição. HertFordShire. Prentice Hall.1988.293 p.

WILTON-JONES, Mark. **Efficient JavaScript.** Dev.Opera, 2006. Disponível em :< https://dev.opera.com/articles/efficient-javascript/> Acessado em: 26/06/2020

ZAKAS, Nicholas C. **Professional JavaScript for Web Developers**. Indianopolis. Wiley Publishing Inc,.2005.646 p.

# APÊNDICE A – VIRTUAL DOM

Para utilizar o React são necessárias no mínimo duas bibliotecas, a biblioteca React que será usada para criar as visualizações, criar os componentes da interface de usuário e o ReactDom que irá renderizar esses componentes no navegador (BANKS & PORCELLO, 2017). A utilização do ReactDom, revela um conceito que ajuda no entendimento das qualidades do React, o conceito de Virtual Dom.

Ao utilizar JavaScript e HTML, faz-se necessário a manipulação do DOM (ver seção 2.8), que permite adicionar a dinâmica das páginas web. Entretanto, a manipulação dos elementos DOM é uma tarefa difícil em termos de conhecimento de JavaScript e demorada em questão de tempo de processamento, além de que sem um utilitário as funções de manipulação do DOM precisam ser refeitas a cada nova aplicação criada (BANKS & PORCELLO, 2017)

As principais lentidões aparecem em duas ações utilizando o DOM, quando deseja-se repintar um elemento ou fazer o chamado refluxo (termo traduzido de maneira literal) do elemento. Repintar é quando o desenvolvedor atribui uma característica que antes não era visível ou retira a visibilidade de alguma característica, sendo uma tarefa dispendiosa por verificar todos os elementos, assim como seu estado anterior para decidir o que será visualizado ou não. O refluxo é mais dispendioso ainda do que o repintar, pois é acionado toda vez que existe uma manipulação da árvore DOM e não são atualizados somente um elemento manipulado, mais seus filhos, ancestrais e sucessores são envolvidos no refluxo, para garantir a integridade de layout entre eles (WILTON-JONES,2006).

Para solucionar os problemas relacionados a manipulação do DOM, o React fornece a possibilidade de delegar para a biblioteca a responsabilidade de atualizar o DOM. Através do React, os desenvolvedores não precisam interagir diretamente com o DOM, mas sim com o Virtual DOM (Banks e Porcelo,2017). O virtual DOM é um padrão de manipulação do DOM, no caso do React ele funciona como um atualizador de estados, conciliando o estado da interface de usuário com o “DOM Real”, garantindo que a atualização ocorra. O motivo da atualização pelo virtual DOM ser mais rápida é pelo seu mecanismo de análise de mudanças, o React consulta a árvore do virtual DOM antes da renderização e depois da nova renderização, calculando as diferenças e a partir desse cálculo o próprio React verifica quais invocações de

métodos do DOM são necessárias para efetuar a atualização, otimizando o processo de manipulação do “DOM real”(STEFANOV,2016).

O maior ganho com o uso do Virtual Dom através do React é a abstração das operações realizadas com DOM, não necessitando que o desenvolvedor se preocupe em manipulá-lo para construir suas aplicações (REACT,2020). O Virtual DOM é composto de elementos React, que são essencialmente objetos JavaScript, atribuindo o seu fator de velocidade ao renderizar componentes em detrimento do “DOM real”, pois é mais fácil atualizar objetos JavaScript do que atualizar elemento do “DOM real” (BANKS & PORCELLO, 2017).

# APÊNDICE B – COMPONENTES DETALHADOS

Na refatoração das aplicações efetuada na seção 5.1 apenas 17 componentes da presente solução foram utilizados, logo, observou-se a necessidade de detalhar os outros componentes que integram a solução proposta. Na refatoração utilizou-se os componentes Message, CheckBox, Field, Notification, Button, TextArea, FielContent, FileButton, Columns, Column, Message, FieldButton, Table, TableQuery, Title, Card e Input.

O presente apêndice apresenta uma breve descrição do componente e duas tabelas, uma contendo as diminuições de tags e linhas HTML ao utilizar o componente e outra tabela as informações referentes a cada atributo de props que o componente possui e uma breve descrição desses atributos.

Na **Tabela *14*** são apresentas a legenda das tabelas do Apêndice. Observa-se que alguns componentes possuem o atributo “itens” e esse atributo é um array de Objetos que teve seu tipo definido pelo Flow na presente solução, logo, os componentes com esse atributo itens possuem uma tabela descritiva a mais.

**Tabela 14** - Legenda das tabelas Apêndice B

| Sigla | Descrição |
|---|---|
| TO | Total de tags do código original |
| TC | Total de tags ao usar o componente |
| LHO | Total de linhas HTML do código original |
| LHR | Total de linhas HTML ao usar o componente |
| %LH | Porcentagem de redução do código HTML |
| %T | Porcentagem de redução do número de tags |

**Fonte**: O próprio autor

1. Box:

O componente Box é inspirado no elemento de igual nome do Bulma (do grupo Elements). Esse componente funciona como um container, onde apenas serve como base para suportar os dados dos seus componentes filhos, logo a redução de tags dele é 0%, pois a quantidade de tag é equivalente ao usar box original (sem a solução), apenas troca-se   por <Box> </Box>.

**Tabela 15** - Métricas Box

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Box | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 16** - Props Box

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Box |
| children | React.Node | Filho do componente Box, conteúdo entre <Box> </Box> |
| custom | Object | Objeto para customização via JSX Spread |

**Fonte**: O próprio autor

2. BreadCrumb

O componente BreadCrumb permite a criação de diversas abas e essas abas permitem a utilização de links, eventos, ícones entre outras possibilidades. Observa-se que no exemplo utilizado para a criação das tabelas abaixo o BreadCrumb possuía 4 itens.

**Tabela 17** - Métricas BreadCrumb

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| BreadCrumb | 20 | 1 | 95 | 10 | 1 | 90 |

**Fonte**: O próprio autor

**Tabela 18** - Props BreadCrumb

| Props | Default | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente BreadCrumb, tag <nav> principal |
| itens | Array<BreadCrumbItem> | Itens que compõem as abas do BreadCrumb |

**Fonte**: O próprio autor

**Tabela 19** - BreadCrumbItem

| BreadCrumbItem | Atributo | Tipo |
|---|---|---|
| | key | object |
| | definition | string |
| | href | string |
| | value | object |
| | onClick | function |
| | custom | object |

**Fonte**: O próprio autor,2020

3. ButtonList

O componente ButtonList funciona como um container para os componentes Button, seu uso não é obrigatório, entretanto é uma boa forma de adicionar espaçamento e organização ao se utilizar diversos botões.

**Tabela 20** - Métricas ButtonList

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| ButtonList | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 21** - Props ButtonList

| Props | Default | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente ButtonList |
| children | React.Node | Filho do componente ButtonList conteúdo entre ButtonList ButtonList |
| custom | Object | Objeto para customização via JSX Spread |

**Fonte**: O próprio autor

4. Container

O componente Container, como o nome já indica, é um container que aceita qualquer tipo de elemento como filho. Assim como os outros componentes que atuam como tipos de container, ele não possui uma redução significativa do número de tags, por ser equivalente em quantidade a implementação original.

**Tabela 22** - Métricas Container

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Container | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 23** - Props Container

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Container |
| children | React.Node | Filho do componente Box, conteúdo entre <Container></Container> |

**Fonte**: O próprio autor

5. Content

Componente que apresenta um container que suporta as tags HTML e que gera um resultado gráfica exatamente conforme o código HTML contido dentro dele. Por ser um container, as diminuições não são significativas.

**Tabela 24** - Métricas Content

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Content | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 25** - Props Content

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Content |
| children | React.Node | Filho do componente Box, conteúdo entre <Content></Content> |

**Fonte**: O próprio autor

6. Control

O componente Control funciona como um embrulho para outros componentes de controle, como label e input por exemplo. Ao se utilizar o Control, o objetivo é manter um espaçamento padrão entre os elementos de controle. O componente Control também funciona como container, logo sua redução de código torna-se não significativa.

**Tabela 26** - Métricas Control

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Control | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 27** - Props Control

| Props | type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Control |
| children | React.Node | Filho do componente Box, conteúdo entre<Contro>l</Control> |

**Fonte**: O próprio autor

7. DropDown

O componente DropDown é um componente do tipo caixa de combinação que fornece diversos itens para seleção. Por ser um componente com diversos itens é possível diminuir consideravelmente o código HTML, muito auxiliado pelo fator de interação presente nele (ver seção 4.1.1.2). Observa-se que o nome DropDown está associado a ideia de um menu que “abre” e “fecha” para a exibição de seus itens.

**Tabela 28** - Métricas DropDown

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| DropDown | 27 | 2 | 92.59259 | 30 | 1 | 96.66667 |

**Fonte**: O próprio autor

**Tabela 29** - Props DropDown

| Props | Default | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente DropDown |
| itens | Array<DropDownItem> | Itens do DropDown, cada elemento selecionável que ele possui |
| onClick | function | Evento ao clicar no primeiro item do componente DropDown |

**Fonte**: O próprio autor

**Tabela 30** - DropDownItem

| DropDownItem | Atributo | Tipo |
|---|---|---|
| | link | boolean |
| | onClick | function |
| | value | object |
| | icon_name | string |
| | href | string |
| | custom | object |
| | definition | string |

**Fonte**: O próprio autor,2020

8. Footer

O componente Footer funciona como um container para o rodapé de páginas web e igual aos containers antes descritos no presente apêndice, ele não possui um reuso de software significante.

**Tabela 31** - Métricas Footer

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Footer | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 32** - Props Footer

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Footer |
| children | React.Node | Filho do componente Box, conteúdo entre <Footer> </Footer> |

**Fonte**: O próprio autor

9. Hero

O componente Hero fornece uma espécie de Banner com um título principal e um subtítulo. O componente apresentou um considerável reuso de software pois utiliza da substituição de tags, de código HTML por informações através de props, onde as tags necessárias para título e subtítulo já são pré-definidas.

**Tabela 33** - Métricas Hero

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Hero | 10 | 1 | 90 | 12 | 1 | 91.66667 |

**Fonte**: O próprio autor

**Tabela 34** - Props Hero

| Props | Type | Descrição |
|---|---|---|
| title | string | O título principal(h1) do componente Hero |
| subtitle | string | O subtítulo(h2) do componente Hero |
| definition | string | Define o estilo do componente Hero |
| container_definition | string | Define o estilo do container dentro de hero_body |
| title_definition | string | Define o estilo para o título |
| subtitle_definition | string | Define o estilo para o subtítulo |

**Fonte**: O próprio autor

10. HeroThreeParts

O componente HeroThreeParts é um componente utilizado para criação de um Hero mais complexo que é dividido em 3 partes que são head, body e footer. Essas três partes são montadas a partir dos filhos de HeroThreeParts que devem ser três, onde o primeiro filho será o header do componente, o segundo o body e o terceiro o footer.

**Tabela 35** - Métricas HeroThreeParts

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| HeroThreeParts | 78 | 11 | 85.89744 | 69 | 8 | 88.4058 |

**Fonte**: O próprio autor

**Tabela 36** - Props HeroThreeParts

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo para o componente HeroThreeParts |
| children | React.Node | Componentes filhos, que devem ser três |
| head_definition | string | Define o estilo do hero-head do HeroThreeParts |
| body_definition | string | Define o estilo do hero-body do HeroThreeParts |
| foot_definition | string | Define o estilo do hero-foot do HeroThreeParts |

**Fonte**: O próprio autor

11. Icon

O componente Icon renderiza um ícone da tela.

**Tabela 37** - Métricas Icon

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Icon | 6 | 1 | 83.33333 | 4 | 1 | 75 |

**Fonte**: O próprio autor

**Tabela 38** - Props Icon

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo para o componente Icon |
| right | boolean | Define que Icon ficará ao lado direito ao ser filho de outro componente |
| onClick | function | Evento ao clicar no ícone |
| icon_name | string | Nome do componente Icon conforme Font Awesome |
| children | React.Node | Filho do componente Icon |

**Fonte**: O próprio autor

12. Image

Componente que trata da renderização de imagem.

**Tabela 39** - Métricas Image

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Image | 3 | 1 | 66.66667 | 3 | 1 | 66.66667 |

**Fonte**: O próprio autor

**Tabela 40** - Props Image

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo para a tag <figure> do componente Image |
| image_definition | string | Define o estilo para a tag img do componente Image |
| src | string | Localização da imagem (URL ou caminho de pasta) |
| onClick | function | Evento ao clicar na imagem |

**Fonte**: O próprio autor

13. Input

Componente campo de texto que recebe dados do usuário por digitação.

**Tabela 41** - Métricas Input

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Input | 5 | 1 | 80 | 5 | 1 | 80 |

**Fonte**: O próprio autor

**Tabela 42** - Props Input

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Input |
| type | string | Define o tipo de Input se é texto, número, checkbox e etc. |
| placeholder | string | Localização da imagem (URL ou caminho de pasta) |
| value | string | Define o valor que aparecerá no Input |
| readonly | boolean | [true] = Componente é apenas para leitura |
| disabled | boolean | [true] = Componente é desabilitado |
| onChangeEvent | function | Evento disparado em mudanças no valor de Input |
| custom | Object | Objeto para customização via JSX Spread |

**Fonte**: O próprio autor

## 14. Level

Componente de layout que permite dividir a estrutura da página web horizontalmente.

**Tabela 43** - Level métricas

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Level | 43 | 39 | 9.302326 | 28 | 24 | 14.28571 |

**Fonte**: O próprio autor

**Tabela 44** - Props Level

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Level |
| itens | Array<LevelItem> \| Array<MediaObjectItem> | Define os itens do Level, o código de cada item do componente Level |

**Fonte**: O próprio autor

**Tabela 45** - LevelItem

| LevelItem | Atributo | Tipo |
|---|---|---|
| | key | object |
| | definition | string |
| | onClick | function |
| | href | string |
| | custom | object |
| | left | boolean |
| | right | boolean |
| | value | object |

**Fonte**: O próprio autor

15. MediaObject

Componente indicado para elementos aninhados e repetidos, como comentários de redes sociais por exemplo.

**Tabela 46** - Métricas MediaObject

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| MediaObject | 41 | 6 | 85.36585 | 32 | 7 | 78.125 |

**Fonte**: O próprio autor

**Tabela 47** - Props MediaObject

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente MediaObject |
| image_definition | string | Define o estilo da tag img do componente Image de MediaObject |
| figure_definition | string | Define o estilo da tag figure do componente Image de MediaObject |
| content_definitio n | string | Define o estilo do componente Content de MediaObject |
| children | React.Node | Componente filho de MediaObject |
| src | string | Define a localização da imagem exibida no componente |
| right | React.Node | [right != undefined] então será renderizado no lado direito o componente informado em right ao invés de um botão delete |
| itens | Array<MediaObjectIt> | Os itens presentes no MediaObject |
| level_definition | string | Define o estilo da do componente Level de Level de MediaObject |
| left_definition | string | Define o estilo da media-left de Level do componente MediaObject |
| right_definition | string | Define o estilo da tag media-right de Level do componente MediaObject |
| content_definitio n | string | Define o estilo do componente Content de MediaObject |
| onClickDelete | function | Define o evento disparado ao clicar no botão delete de MediaObject |

**Fonte**: O próprio autor

**Tabela 48** - MediaObjectItem

| MediaObject | Atributo | Tipo |
|---|---|---|
| | key | number |
| | left | boolean |
| | onClick | function |
| | value | object |

**Fonte**: O próprio autor,2020

## 16. Menu e MenuItem

Componente que cria uma estrutura de menu no formato vertical.Observa-se que o exemplo utilizado em Item foi construído através de um menu com 3 itens

**Tabela 49** - Métricas Menu e MenuItem

| Componentes | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Menu e MenuItem | 68 | 8 | 88.23529 | 28 | 24 | 14.28571 |

**Fonte**: O próprio autor

**Tabela 50** - Props Menu e MenuItem

| Componente | Props | Type | Descrição |
|---|---|---|---|
| Menu | definition | string | Define o estilo do componente Level |
| Menu | children | children | Componente filho de Menu, no caso, deve ser MenuItem |
| MenuItem | itens | Array<MenuItemItem> | Define os itens do Level, o código de cada item do MenuItem |
| MenuItem | label | string | Nome destacado de cada MenuItem |

**Fonte**: O próprio autor

**Tabela 51** - MenuItemItem

| MenuItemItem | Atributo | Tipo |
|---|---|---|
| | onClick | function |
| | value | object |
| | definition | string |
| | sub_itens | Array<{definition?:string, value?: Object, onClick: ()=> void}> |

**Fonte**: O próprio autor

## 17. NavigationBar

Componente que fornece a possibilidade de criar uma barra de navegação na página web, com imagem no lado esquerdo e componentes iniciais e finais.

**Tabela 52** - Métricas NavigationBar

| Componentes | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| NagivationBar | 39 | 3 | 92.30769 | 32 | 11 | 65.625 |

**Fonte**: O próprio autor

**Tabela 53** - Props NavigationBar

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente NavigationBar |
| link | string | link de redirecionamento ao clicar na foto do Brand de NavigationBar |
| src | string | localização da foto renderizada no Brand de NavigationBar |
| width_brand | number | Largura do Brand de NavigationBar |
| alt | string | atributo alt da tag <img> |
| height_brand | number | Altura do Brand de NavigationBar |
| itens_start | Array<NavigationBarItem> | Itens que pertence ao começo da NavigationBar (lado esquerdo) |
| itens_end | Array<NavigationBarItem> | Itens que pertence ao fim da NavigationBar (lado direito |

**Fonte**: O próprio autor

**Tabela 54** - NavigationBarItem

| NavigationBarItem | Atributo | Tipo |
|---|---|---|
| | definition | string |
| | onClick | function |
| | value | object |
| | itens_start | {definition?: string, onClick?: function, value?:string} |
| | itens_end | {definition?: string, onClick?: function, value?:string, custom?: object} |

**Fonte**: O próprio autor.

## 18. Pagination

Componente que permite a paginação da interface gráfica na web, permitindo redirecionamento e eventos aos clicar nos números de páginas e links next e previous. Os links next e previous na realidade podem possuir qualquer nome, tratam-se uma estrutura pré-definida de dois links que um disparada a ação ao clicar em “voltar” (previous) e outro a ação de “avançar” (next)

**Tabela 55** - Métricas Pagination

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Pagination | 20 | 1 | 95 | 15 | 1 | 93.33333 |

**Fonte**: O próprio autor

**Tabela 56** - Props Pagination

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Pagination |
| href_previous | string | link para redirecionamento da tag <a> previous |
| href_next | string | link para redirecionamento da tag <a> next |
| next_disabled | boolean | link next desativado |
| next_name | string | nome da tag <a> next |
| previous_disabled | boolean | link previous desativado |
| itens | Array<PaginationItem> | números de páginas do componente Pagination |
| previous_name | string | nome da tag <a> previous |

**Fonte**: O próprio autor

**Tabela 57** - PaginationItem

| PaginationItem | Atributo | Tipo |
|---|---|---|
| | definition | string |
| | value | object |
| | onClick | function |
| | disabled | boolean |
| | href | string |

**Fonte**: O próprio autor,2020

## 19. Panel

Componente de para composição de elementos de controle. O componente Panel fornece a opção de criar uma série de blocos no seu corpo, um Field para pesquisa dos blocos, um componente Tabs para dividir os blocos por tipo e um botão para eventuais ações.

**Tabela 58** - Métricas Panel

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Panel | 68 | 4 | 94.11765 | 65 | 3 | 95.38462 |

**Fonte**: O próprio autor

**Tabela 59** - Props Panel

| Props | Type | Descrição |
|---|---|---|
| header | string | Nome que aparece em destaque no Header do componente Panel |
| itens_tabs | Array<PanelItemTabs> | Itens que formam as abas do componente Panel |
| itens_blocks | Array<PanelItemItem> | Itens que formam os blocos(corpo) do componente Panel |
| definition | string | Define o estilo do componente Panel |
| search_definition | string | Define o estilo do componente Field de pesquisa |
| search_placeholder | string | Define o placeholder do componente Field de pesquisa |
| search_icon_definitio n | string | Define a definição do ícone do componente Field de pesquisa |
| onChangeSearch | function | Evento disparado toda vez que mudar o texto dentro do Field de pesquisa |

**Fonte**: O próprio autor

**Tabela 60** – PanelItemItem

| PanelItemItem | Atributo | Tipo |
|---|---|---|
| | onClick | function |
| | icon_name | string |
| | icon_definition | string |
| | value | object |
| | href | string |

**Fonte**: O próprio autor,2020

**Tabela 61** - PanelItemTabs

| PanelItemTabs | Atributo | Tipo |
|---|---|---|
| | onClick | function |
| | value | string |
| | href | string |
| | icon_definition | string |
| | icon_name | string |

**Fonte**: O próprio autor,2020

## 20. ProgressBar

O componente ProgressBar fornece uma barra de progresso que pode ser utilizada para informar a evolução de alguma demanda através da interface gráfica.

**Tabela 62** - Métricas ProgressBar

| Componentes | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| ProgressBar | 1 | 1 | 0 | 1 | 1 | 0 |

**Fonte**: O próprio autor

**Tabela 63** - Props ProgressBar

| Props | Type | Descrição |
|---|---|---|
| onChange | function | Evento disparado sempre que mudar o atributo value de <progress> |
| max | number | Valor máximo que o atributo value de <progress> pode atingir |
| value | string | Valor de ProgressBar |
| definition | string | Define o estilo do componente ProgressBar |

**Fonte**: O próprio autor

## 21. Radio

Componente que cria uma série de botões de opções, que possuem natureza binária (marcados ou desmarcados).

**Tabela 64** - Métricas Radio

| Componentes | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Radio | 8 | 1 | 87.5 | 10 | 1 | 90 |

**Fonte**: O próprio autor

**Tabela 65** - Props Radio

| Props | Type | Descrição |
|---|---|---|
| itens | Array<RadioItem> | Itens do componente Radio |
| name | string | Atributo name de <input> do Radio |

**Fonte**: O próprio autor

**Tabela 66** - RadioItem

| RadioItem | Atributo | Tipo |
|---|---|---|
| | onClick | function |
| | checked | boolean |
| | custom | Object |
| | value | Object |
| | disabled | boolean |

**Fonte**: O próprio autor

22. SearchTable

Esse componente fornece a possibilidade de criar uma tabela com pesquisa dinâmica de dados através da sua construção utilizando diversos componentes da presente solução.

**Tabela 67** - Métricas SearchTable

| Componentes | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| SearchTable | 124 | 2 | 98.3871 | 78 | 10 | 87.17949 |

**Fonte**: O próprio autor

**Tabela 68** - Props SearchTable

| Props | Type | Descrição |
|---|---|---|
| onChange | function | Evento disparado ao mudar o valor do Field de pesquisa |
| label | string | Rótulo do componente |
| control_definition | string | Define o estilo do componente Control do Field de pesquisa |
| placeholder | string | Define o placeholder do Field de pesquisa |
| icon_name | string | Nome do Icon do Field de pesquisa |
| icon_definition | string | Define o estilo do Icon do Field de pesquisa |
| input_definition | string | Define o estilo do Input do Field de pesquisa |
| table_definition | string | Define o estilo da Table com o resultado da pesquisa |
| itens_header | Array<TableItem> | Itens do cabeçalho da Table com o resultado da pesquisa |
| itens_body | Array<Object> | Itens do corpo da Table com o resultado da pesquisa |
| header_definition | string | Define o estilo do header da Table com o resultado da pesquisa |
| itens_footer | Array<TableItem> | Itens do rodapé da Table com o resultado da pesquisa |
| button_definition | string | Define o estilo do Button de SearchTable |
| onClick | function | Evento disparado ao clicar no Button de SearchTable |
| value | string | Valor do Field de pesquisa |
| onClickRow | function | Evento disparado ao clicar em alguma linha da Table |

**Fonte**: O próprio autor

23. Section

Componente que divide a página web em seções,

**Tabela 69** - Métricas Section

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Section | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 70** - Props Section

| Props | Default | Descrição |
|---|---|---|
| definition | X | Define o estilo do componente Section |
| children | X | Filho do componente Section conteúdo entre <Section> </Section> |

**Fonte**: O próprio autor

24. Select

O componente Select funciona como uma caixa de combinação que apresenta diversos itens para seleção.

**Tabela 71** - Métricas Select

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Select | 8 | 1 | 87.5 | 5 | 1 | 80 |

**Fonte**: O próprio autor

**Tabela 72** - Props Select

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Select |
| onChange | função | Dispara um evento toda vez que um item for selecionado |
| multiple_size | number | Define um tamanho de linhas para o Select |
| itens | Array<SelectItem> | Define os itens do Select |
| custom | Object | Objeto para customização via JSX Spread |

**Fonte**: O próprio autor

**Tabela 73** - SelectItem

| SelectItem | Atributo | Tipo |
|---|---|---|
| | value | Object |

**Fonte**: O próprio autor

25. Subtitle

Fornece uma opção para criação de um subtítulo

**Tabela 74** - Métricas Subtitle

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Subtitle | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 75** - Props Subtitle

| Props | Type | Descrição |
|---|---|---|
| definition | number | Define o estilo do componente Subtitle |
| p | boolean | [true] Então o subtítulo será construído através da tag <p> e não da tag  |
| children | React.Node | Filho do componente SubTitle |

**Fonte**: O próprio autor

26. Tabs

Componente para criar um menu dividido em abas.

**Tabela 76** - Métricas Tabs

| Componente | TO | TC | T% | LHO | LHR | LH% | |
|---|---|---|---|---|---|---|---|
| Tabs | 20 | 1 | 95 | 16 | 1 | 93.75 | |

**Fonte**: O próprio autor

**Tabela 77** - Props Tabs

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Tabs |
| itens | Array<TabItem> | [true] Itens que formam as abas do componente Tabs |

**Fonte**: O próprio autor

**Tabela 78** - TabItem

| TabItem | Atributo | Tipo |
|---|---|---|
| | onClick | function |
| | definition | string |
| | icon_name | string |
| | icon_definition | string |
| | value | string |

**Fonte**: O próprio autor

27. Tag

Componente para criar pequenas “etiquetas” gráficas. Observa-se que no exemplo utilizado o componente Button foi considerado, por isso três tags no código original.

**Tabela 79** - Métricas Tag

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Tag | 3 | 1 | 66.66667 | 3 | 1 | 66.66667 |

**Fonte**: O próprio autor

**Tabela 80** - Props Tag

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Tag |
| children | React.Node | Componente filho de Tag |
| onClickDelete | function | Evento executado ao clicar no botão delete, caso Tag seja delete. |
| delete | boolean | [true] Então Tag possui um Button do tipo delete |

**Fonte**: O próprio autor

28. TagList

Componente que funciona como um container para componentes do tipo Tag

**Tabela 81** - Métricas TagList

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| TagList | 14 | 1 | 92.85714 | 8 | 1 | 87.5 |

**Fonte**: O próprio autor

**Tabela 82** - Props TagList

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente TagList |
| itens | Array<TagItem> | Itens que formam as tags de TagList |

**Fonte**: O próprio autor

**Tabela 83** - TagItem

| TagItem | Atributo | Tipo |
|---|---|---|
| | value | Object |
| | definition | string |
| | onClick | function |
| | custom | Object |

**Fonte**: O próprio autor

29. Tile

Componente que permite a divisão da interface em grades de diferentes tamanhos, ou conforme encontrado na documentação do Bulma, “componentes bidimensionais” (Bulma,2020).

**Tabela 84** - Métricas Tile

| Componente | TO | TC | T% | LHO | LHR | LH% |
|---|---|---|---|---|---|---|
| Tile | 2 | 2 | 0 | 2 | 2 | 0 |

**Fonte**: O próprio autor

**Tabela 85** - Props Tile

| Props | Type | Descrição |
|---|---|---|
| definition | string | Define o estilo do componente Tile |
| children | React.Node | Componente filho de Tile |

**Fonte**: O próprio autor

## APÊNDICE C – ARTIGO SBC

# ASSEMBLE REACT BULMA: UMA BIBLIOTECA DE COMPONENTES REACT INSPIRADOS NO FRAMEWORK CSS BULMA

**Thiago Miklan Moreira**

Graduando em Sistemas de Informação - Universidade Federal de Santa Catarina

**Leandro José Komosinski**

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) - Florianópolis – SC – Brasil

{miklanthiago@gmail.com, leandro.komosinski@ufsc.br}

***Abstract.*** *In Web development, CSS, HTML and JavaScript predominate in the market as solutions for the creation of web systems of the most varied sizes. With the growing adhesion of JavaScript by developers, several frameworks and libraries were created to assist the development of systems with these technologies, when observing that the integration of the three languages to create a web system causes a considerable amount of code repetition. React is a JavaScript library for building graphical interfaces, which seeks to offer developers who use JavaScript a reuse of the codes created for the construction of interfaces. Bulma is a CSS framework that predefines several elements that can be used by JavaScript developers to manipulate the appearance of developed applications. When using Bulma in conjunction with React, a problem was introduced, the large amount of HTML code repeated in the integration of the two technologies, opposing the primary idea of frameworks and libraries to facilitate software reuse. In order to improve software reuse when using React and Bulma together, a library of React components developed specifically for use with the CSS Bulma framework was developed in the present work. At the end of this work, check if it is possible to considerably decrease the HTML code needed to improve the use of React and Bulma.*

***Resumo.*** *No desenvolvimento Web, as linguagens CSS, HTML e JavaScript predominam no mercado como soluções para a criação de sistemas web dos mais variados portes. Com a crescente adesão do JavaScript por parte dos desenvolvedores, diversos frameworks e bibliotecas foram criados para darem auxílio ao desenvolvimento de sistemas com essas tecnologias, ao observar que a integração das três linguagens para criação de um sistema web acarreta em uma quantidade considerável de repetição de código. O React é uma biblioteca JavaScript para construção de interfaces gráficas, que busca oferecer aos desenvolvedores que utilizam JavaScript um reaproveitamento dos códigos criados para a construção das interfaces. O Bulma é um framework CSS que predefine diversos elementos que podem ser utilizados pelos desenvolvedores JavaScript para manipular as aparências das aplicações desenvolvidas. Ao usar o Bulma em conjunto com o React é introduzido um problema, a grande quantidade de código HTML repetido na*

*integração das duas tecnologias, contrapondo a ideia primária dos frameworks e bibliotecas de facilitar o reuso de software.Com o intuito de melhorar o reuso de software ao utilizar o React e o Bulma em conjunto, desenvolveu-se no presente trabalho uma biblioteca open source de componentes React implementados especificamente para utilização com o framework CSS Bulma. Ao finalizar o presente trabalho, constatou-se que é possível diminuir consideravelmente o código HTML necessário fornecendo melhor produtividade ao utilizar React e Bulma.*

## 1. Introdução

As aplicações web são sistemas que operam em navegadores na Internet e são construídos a partir de várias linguagens de programação. Os desenvolvedores de aplicações web, na maioria dos casos, necessitam utilizar pelo menos três linguagens: JavaScript, HTML e CSS.

O uso do JavaScript aumentou consideravelmente nos últimos anos. O JavaScript foi a linguagem com mais repositórios criados na plataforma de versionamento de código GitHub desde o ano de 2011, sendo a linguagem mais utilizada pelos colaboradores da plataforma [Elliott 2018].Uma pesquisa publicada no site StackOverFlow, mostra que em 2018 as três mais populares tecnologias do site foram JavaScript, HTML e CSS [Stackoverflow 2018].

Com o crescimento acentuado dos desenvolvedores de aplicações web, por efeito colateral, desenvolveu-se uma quantidade considerável de frameworks e bibliotecas, que visam facilitar o trabalho dos programadores através principalmente do reuso de software.

O reuso de software é o reaproveitamento de software que já foi feito em detrimento de criar um outro software totalmente novo. As principais vantagens de se ter um alto reuso de software são o custo do software cair, o tempo de produção diminuir e a qualidade aumentar, tudo isso pelo fato dos componentes utilizados serem previamente testados, documentados, mantidos e codificados [Fayad 2016].

Ao conceito de componente, uma variedade de definições é fornecida, no presente trabalho observou-se necessário definir o conceito de componente a partir da visão tradicional dele, abstraindo definições referentes a orientação a objetos, processos e outros. Segundo Pressman (2011, pg 258), componente é definido como: “Componente é um bloco construtivo modular para software de computador”.

A solução desenvolvida no presente trabalho é construída na ideia de programação funcional, através de componentes React funcionais. No paradigma funcional, as funções são constituídas de procedimentos e podem depender de outras funções auxiliares para ajudar a alcançar os seus valores finais, além disso, as funções possuem tipos que classificam os valores envolvidos da função [Wadler 1988].

O React é uma biblioteca JavaScript criada pelo *Facebook* que busca auxiliar na criação de interfaces gráficas de usuário. O React é baseado em componentes e possui uma característica de dinamismo na exibição das interfaces, onde cada componente é renderizado conforme o estado da aplicação [React 2020].

O Bulma é um framework CSS que oferece uma padronização da aparência das páginas Web, permitindo que os programadores consigam criar facilmente uma interface gráfica com um nível aceitável de estética [Bulma 2020]. Optou-se pelo Bulma como framework CSS pela sua simplicidade, responsividade e estética, além da confiança de que o framework pode ser

mais utilizado pelos desenvolvedores e possuir qualidades que se destacam frente a outros frameworks

É comum no desenvolvimento web a integração de frameworks e bibliotecas, o React e o Bulma podem e são usados em conjunto para criação de interfaces gráficas. O principal problema que surge com a integração dessas tecnologias é a grande repetição de código HTML.

No decorrer da pesquisa, analisou-se soluções correlatas ao presente trabalho, com basicamente os mesmos objetivos, de todas as soluções analisadas, duas se destacaram que foram *Brightleaf Elements* e *React Bulma Components*. Da análise realizada sobre as soluções, notou-se fatores negativos quanto ao reuso de software pois par usá-las é necessário muitas tags, muitos componentes e uma dificuldade quanto a complexidade de uso, por necessitar de muitas memorizações de componentes, nomenclaturas entre outros problemas analisados.

Observando a grande quantidade de código HTML produzido pela integração entre React e Bulma, a falta de trabalhos construídos especificamente para essa integração, pela importância que o reuso de software possui no desenvolvimento de aplicações e pelos pontos negativos encontrados nas soluções correlatas, o presente trabalho objetivou criar uma biblioteca de componentes funcionais React chamada Assemble React Bulma que visa facilitar o desenvolvimento ao utilizar React e Bulma, diminuindo em pelo menos 25% das tags HTML necessárias para um componente refatorado.

## 2. Solução Desenvolvida

No presente tópico é apresentado um panorama geral da solução desenvolvida para alcançar os objetivos definidos, onde apresenta-se os requisitos funcionais e não funcionais da solução e alguns aspectos chaves da solução. Os requisitos da solução proposta são expressos na **Tabela 1** e **Tabela 2**.

**Tabela 86** - Requisitos funcionais

| | **Requisitos funcionais** |
|---|---|
| **RF** | **Descrição** |
| 1 | A soluçao deve implementar o conceito de interação, conforme definido na seção 4.2 |
| 2 | Um componente React deve ser criado para cada elemento julgado importante para a solução |
| 3 | Um arquivo .flow deve ser gerado para cada componente React da solução |
| 4 | A solução deve possibilitar averiguação de tipos em tempo de compilação aos usuários |
| 5 | A solução deve impedir que dados de tipos errados sejam usados pelos componentes |

**Fonte**: O próprio autor,2020.

**Tabela 87** - Requisitos não funcionais

| | **Requisitos Não Funcionais** |
|---|---|
| **RNF** | **Descrição** |
| 1 | A nomenclatura da parte estética da solução deve ser igual a fornecida pelo Framework Bulma |
| 2 | Os componentes React da solução precisam ser todos do tipo stateless |
| 3 | Os componentes React da solução devem ser criados através de um modelo pré-definido (conforme especificado na seção 4.1 , seja esse modelo fornecido pelo Bulma ou através da pesquisa prática |
| 4 | Os atributos "props" dos componentes React da solução devem ser padronizados, conforme definido na seção 4.3 |
| 5 | A solução deve funcionar com o framework Bulma na versão 0.9.0 |
| 6 | A solução deve funcionar com a biblioteca React na versão 16.13.1 |
| 7 | A solução deve ser disponibilizada no repositório NPM |
| 8 | A solução deve ter uma documentação em formato Github Wiki disponibilizada |

**Fonte**: O próprio autor,2020

A solução desenvolvida busca gerar um resultado equilibrado entre customização e facilidade no uso, optando por adicionar uma certa customização aos componentes sem adicionar complexidades desnecessárias.

Manteve-se a nomenclatura do Bulma, principais motivos: não precisar aprender uma nova nomenclatura, permitir a compatibilidade com os elementos fornecidos pelo Bulma e maior flexibilidade ao definir os elementos.

Os componentes desenvolvidos na presente solução abrangem os grupos Columns, Layout, Form, Elements e Components. Observa-se que existem adaptações quanto ao conteúdo proposta na documentação do Bulma e a implementação na presente solução. Ao total, **48 componentes** foram desenvolvidos para o presente trabalho.

De todos os aspectos importantes para a presente solução, duas ideias são fundamentais para compreensão da presente solução e de como ela busca diminuir o código gerado na integração React/Bulma, que são a criação de uma estrutura pré-definida para os componentes e a interação para criação de código HTML.

Abaixo, na **Figura 153**,**Figura 154**, e **Figura 155** é apresentado um exemplo de como funcionam os conceitos de estrutura pré-definida e interação para criação de código HTML. Na **Figura 153** é possível observar que existe a criação de uma tabela utilizando HTML, Bulma e poderia ser esse código facilmente componentizado com o React, nota-se no código uma estrutura composta de tags e uma grande repetição de código para criação dos itens (tags <th> e <td>. Na **Figura 154** é mostrado como fazer um resultado gráfico equivalente ao da **Figura 153** com uma quantidade inferior de tags envolvidas (aplicando-se a solução), onde o código HTML do header e body são substituídos pelas variáveis itens_body e itens_header. O conceito de estrutura pré-definida, é basicamente selecionar uma estrutura genérica, no caso de Table essa estrutura foi selecionada da documentação do Bulma, e a partir dessa estrutura desenvolver o componente. O conceito de interação para criação de código HTML trata de transformar uma grande repetição de código através de interação, no caso do exemplo, é substituir os itens (<th> e <td>) por uma simples variável escrita em JavaScript que define os valores de cada item, onde o significado de interação está justamente em internamente o componente Table interagir, fazer

um loop para criar esses itens dinamicamente através dessa variável JavaScript, não necessitando da informação manual de cada item.

**Figura 153** - Código sem a solução

```
<table className="table is-bordered is-striped is-narrow is-hoverable">
    <thead>
        <th>Nome</th>
        <th>Year</th>
        <th>Email</th>
    </thead>

    <tbody>
        <tr>
            <td>Walter White</td>
            <td>2013</td>
            <td>@heisenberg</td>
        </tr>
        <tr>
            <td>Jackson Teller</td>
            <td>2014</td>
            <td>@jax</td>
        </tr>

        <tr>
            <td>Anthony Soprano</td>
            <td>2007</td>
            <td>@tony</td>
        </tr>
    </tbody>
</table>
```

**Fonte**: O próprio autor,2020

**Figura 154** - Código com a aplicação da solução

```
let  itens_header = [{value:"Nome"},{value:"Year"},{value:"Email"}]
let itens_body = [  {Nome:"Walter White",Year:"2013", Email:"@heisenberg"},
                    {Nome:"Jackson Teller", Year:"2014",Email:"@jax"},
                    {Nome:"Anthony Soprano",Year:"2007",Email:"@tony"}
                ];

<Table definition="is-bordered is-striped is-narrow is-hoverable" itens_body={itens_body} itens_header={itens_header}/>
```

**Fonte**: O próprio autor,2020

**Figura 155** - Resultado gráfico da Tabela

| Nome | Year | Email |
|---|---|---|
| Walter White | 2013 | @heisenberg |
| Jackson Teller | 2014 | @jax |
| Anthony Soprano | 2007 | @tony |

**Fonte**: O próprio autor,2020

## 3. Resultados Obtidos

O presente capítulo trata dos resultados obtidos com a solução proposta mostrando as métricas, aplicações da solução assim como as melhorias observadas a partir do seu uso prático.

Uma das formas de atestar a resolução fornecida pela presente solução foi através do *refactoring* de uma série de aplicações utilizadas na disciplina Programação para Web (INE5646), do curso de Sistemas de Informação da UFSC, no semestre de 2019/2. As aplicações que receberam o *refactoring* são sistemas com baixa complexidade que utilizam a tecnologia React e Bulma para criar suas interfaces gráficas, entretanto não utilizam outras soluções para facilitar o desenvolvimento. Sendo assim, realizou-se o *refactoring* de 3 aplicações e analisou-se a diferença entre os códigos das aplicações originais (construídas sem a solução proposta) e os códigos das aplicações resultantes.

As métricas utilizadas para análise do refactoring são baseadas em linhas e tags. A métrica através da quantidade de linhas possui uma grande subjetividade, pois essa quantidade muitas vezes pode sofrer mudanças pelo modo como o desenvolvedor organiza seu código, portanto optou-se por adicionar também a métrica de quantidade de tags. Observa-se que será utilizado também a métrica de linhas totais, tanto com código HTML quanto com código JavaScript, para a análise do percentual total de redução do código.

As três aplicações que passaram pelo refactoring chamam-se App Lembretes, App Vulcões e App Compra e Vende. Os resultados obtidos podem ser observados a seguir sendo **Tabela 4** as legendas e as tabelas **5,6 e 7** os resultados obtidos.

**Tabela 4** – Legenda Métricas Refactoring

| Sigla | Descrição |
|---|---|
| L O | Total de linhas do código original |
| L R | Total de linhas do código refatorado |
| TO | Total de tags do código original |
| TR | Total de tags do código refatorado |
| LHO | Total de linhas HTML do código original |
| LHR | Total de linhas HTML do código refatorado |
| %LH | Porcentagem de redução do código HTML |
| %LT | Porcentagem de redução do código total |
| %T | Porcentagem de redução do número de tags |

**Fonte**: O próprio autor, 2020

**Tabela 5** - Métrica refactoring App Lembretes

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| App | 93 | 89 | 11 | 7 | 22 | 17 | 4.3 | 36.36 | 22.73 |
| Login | 233 | 206 | 45 | 16 | 56 | 25 | 11.59 | 64.44 | 55.36 |
| MostraLembretes | 47 | 47 | 11 | 8 | 14 | 13 | 0 | 27.27 | 7.14 |
| PublicaLembrete | 56 | 55 | 9 | 4 | 7 | 4 | 1.79 | 55.56 | 42.86 |

**Fonte**: O próprio autor, 2020

**Tabela 6** - Métricas refactoring App Vulcao

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| CadastraVulcao | 306 | 282 | 41 | 13 | 72 | 47 | 7.84 | 68.29 | 34.72 |

**Fonte**: O próprio autor, 2020

**Tabela 7** - Métricas refactoring de App Compra e Vende

| Componente React | LO | LR | TO | TR | LH0 | LHR | %LT | %T | %LH |
|---|---|---|---|---|---|---|---|---|---|
| App | 24 | 20 | 12 | 8 | 13 | 8 | 16.67 | 33.33 | 38.46 |
| FazPedido | 248 | 210 | 54 | 16 | 114 | 55 | 15.32 | 70.37 | 51.75 |
| TabelaProdutos | 38 | 27 | 24 | 2 | 20 | 4 | 28.95 | 91.67 | 80 |
| TabelaProdutosProcessados | 34 | 25 | 26 | 1 | 19 | 1 | 26.47 | 96.15 | 94.74 |
| MostraPedidosProcessados | 66 | 52 | 32 | 10 | 32 | 10 | 21.21 | 68.75 | 68.75 |
| VerificaPedido | 136 | 115 | 36 | 13 | 41 | 20 | 15.44 | 63.89 | 51.22 |
| App(App Venda) | 25 | 21 | 13 | 9 | 14 | 9 | 16 | 30.77 | 35.71 |
| Processa Pedido | 47 | 43 | 13 | 4 | 19 | 15 | 8.51 | 69.23 | 21.05 |
| TabelaPedidos | 35 | 30 | 26 | 6 | 19 | 7 | 14.29 | 76.92 | 63.16 |
| TabelaProdutosProcessados | 41 | 29 | 29 | 6 | 26 | 7 | 29.27 | 79.31 | 73.08 |
| VeificaPedidos | 221 | 206 | 26 | 15 | 42 | 35 | 6.79 | 42.31 | 16.67 |

**Fonte**: O próprio autor, 2020

## 4.Conclusão

O presente trabalho buscou uma forma de diminuir a quantidade de código necessário ao utilizar a biblioteca React e Bulma para se desenvolver aplicações Web, através de uma biblioteca de componentes *open source* criada especificamente para a integração React/Bulma.

Com o desenvolvimento do trabalho foi possível concluir que a solução diminui de maneira considerável a quantidade de código HTML necessário para construção de interfaces gráficas utilizando React e Bulma, permitindo criar soluções completas com uma menor digitação de código.

Apesar da redução de código, alguns pontos ao decorrer do desenvolvimento foram considerados limitadores e negativos da presente solução.

O principal ponto negativo é a dificuldade de prever quais componentes poderiam ser úteis ou desejados pelos usuários da solução, uma dificuldade de expansão da biblioteca de

componentes a partir da possível necessidade dos usuários, que podem ser dos mais variados tipos.

A curva de aprendizado do React e as tecnologias envolvidas no seu uso podem ser fatores negativos ao aprendizado da presente solução, essa conclusão está suportada pela experiência do autor ao utilizar as tecnologias diariamente sem possuir experiência prévia. No desenvolvimento da presente solução, diversos foram os problemas no aprendizado de tecnologias e conceitos, mas de todas as negativas no aprendizado, o React se torna o problema mais complexo, por funcionar através de um paradigma não tão comum (programação reativa), a tecnologia mostra uma curva de aprendizado grande e conceitos implícitos que só foram entendidos mediante a grande pesquisa prática. Observando as dificuldades antes citadas, é bem provável que usuários que não possuam conhecimentos básicos em Node JS, React, React *Hooks* e gerência de estado passem por alguns problemas no uso da presente solução.

Durante o desenvolvimento alguns trabalhos futuros foram pensados como possíveis melhorias para a presente solução. Os dois considerados mais interessantes pelo autor são a criação de uma biblioteca focada na parte comportamental do React/Bulma e a criação de uma biblioteca de componentes dependentes mais complexos a partir dos componentes implementados no presente trabalho.

A presente solução por ser focada especificamente na redução do código HTML, acaba por não entregar uma grande redução do código geral de um componente, isto é, da parte comportamental de um componente. Observando esse problema, como trabalho futuro sugere-se uma expansão do presente trabalho que lide com o código que não seja HTML de um componente React/Bulma, como por exemplo a utilização dos *hooks* e eventos do componente.

Uma outra sugestão de trabalho futuro é a elaboração de componentes React/Bulma focados somente em componentes dependentes de complexidade elevada. Conforme observado na refatoração apresentado na seção 5 os componentes que são mais complexos, que abstraem uma grande quantidade de componentes, são os que mais apresentam reuso de software, logo, recomenda-se um trabalho que o objetivo seja a elaboração de componentes complexos de diversos tipos e domínios de uso que utilizem como base os implementados no presente trabalho.

O resultado final do presente trabalho conta com a biblioteca pronta para o uso e disponível para download no repositório NPM, onde para encontrar o pacote dela deve-se informar as palavras chaves “assemble-react-bulma”. Pontua-se que é uma biblioteca *open source* e oferecida com a licença **MIT**.

## 5.Referências

BULMA**. Bulma:** free, open source, and modern CSS framework based on Flexbox.2020. Disponível em :<https://bulma.io/>. Acesso em: 30 jan. 2020.

GITHUB. **The State of the Octoverse**.2019. Disponível em: <https://octoverse.github.com/>. Acesso em: 20 out. de 2019.

FAYAD, Mohamed; FLOOD, Charles. Software Reuse Knowledge Map. **Ijars: International Journal of Engineering.** San Jose, p. 1-26. set. 2016. Disponível em:<https://www.researchgate.net/publication/309271508_Software_Reuse_Knowledge_Map>. Acesso em: 24/01/2020

PRESSMAN, Roger S. **Engenharia de Software:** Uma Abordagem Profissional. 7º edição. Porto Alegre: AMGH Editora LTDA.2011.180 p.

REACT**. React:** A JavaScript library for building user interfaces. 2020. Disponível em: <https://pt-br.reactjs.org/>. Acessado em: 30/01/2020

STACKOVERFLOW. **Most Popular Technologies**,2020. Disponível em: < https://insights.stackoverflow.com/survey/2020#most-popular-technologies > Acesso em: 16 de nov. de 2020.

WADLER, Philip; BIRD, Richard. **Introduction to functional programming**. 1º edição. HertFordShire. Prentice Hall.1988.293 p.

## APÊNDICE D – CÓDIGO FONTE

O código-fonte da presente solução, está disponível no link **https://github.com/ThiagoMiklan/Biblioteca** publicamente e distribuído sobre a licença **MIT.**