UNIVERSIDADE DE TAUBATÉ
DIEGO RODRIGO SILVA FERREIRA
GABRIEL D'AVILA LOURENÇO CARDOSO

HELIOS UI: biblioteca de componentes para React JS

TAUBATÉ – SP
2020

DIEGO RODRIGO SILVA FERREIRA
GABRIEL D'AVILA LOURENÇO CARDOSO

HELIOS UI: biblioteca de componentes para React JS

Trabalho de conclusão de curso apresentado para obtenção do Certificado de Graduação pelo Curso de Engenharia da Computação do Departamento de Informática da Universidade de Taubaté,
Orientador: Prof. Me.Dawilmar Guimarães de Araújo.

TAUBATÉ - SP
2020

Grupo Especial de Tratamento da Informação - GETI
Sistema Integrado de Bibliotecas – SIBi
Universidade de Taubaté - Unitau

F383h Ferreira, Diego Rodrigo Silva
HELIOS UI : biblioteca de componentes para React JS / Diego Rodrigo Silva Ferreira, Gabriel D'Avila Lourenço Cardoso. -- 2020.
56 f. : il.

Monografia (graduação) – Universidade de Taubaté, Departamento de Informática, 2020.
Orientação: Prof. Me. Dawilmar Guimarães de Araújo, Departamento de Informática.

1. Biblioteca de componentes. 2. JavaScript. 3. React. 4. TypeScript. I. Cardoso, Gabriel D'Avila Lourenço II.Universidade de Taubaté. Departamento de Informática. Graduação em Engenharia de Computação. III. Titulo.

CDD – 005.133

Ficha catalográfica elaborada pela Bibliotecária Luciene Lopes - CRB 8/5275

DIEGO RODRIGO SILVA FERREIRA

GABRIEL D'AVILA LOURENÇO CARDOSO

HELIOS UI: biblioteca de componentes para React JS

Monografia apresentada como requisito de conclusão do Curso de Engenharia de Computação do Departamento de Informática da Universidade de Taubaté.

Área de concentração: Computação

Orientador: Prof. Me. Dawilmar Guimarães Araújo

Data: 25/12/2020

Resultado: Aprovado

BANCA EXAMINADORA

Prof .Me. Dawilmar Guimarães Araújo

Universidade de Taubaté

Prof .Dr. Eduardo Hidenori Enari

Universidade de Taubaté

Prof .Me. Antonio Esio Marcondes Salgado

Universidade de Taubaté

Dedico este trabalho a Gustavo Marques Martins, por ser o mentor que nos guiou para o caminho de estudos e conhecimentos necessários para chegar aqui.

AGRADECIMENTOS

Ao CTO Gustavo Marques Martins, por ser o mentor que nos guiou para os estudos necessários para chegarmos até aqui.

Ao CTO Hugo Grochau, por nos orientar e tornar possível o desenvolvimento desse projeto.

Ao Prof. Me. Dawilmar Guimarães de Araújo, pela habilidade com que orientou nosso trabalho.

RESUMO

O trabalho apresenta o estudo e desenvolvimento de uma biblioteca de componentes para ser utilizada juntamente ao React, uma das bibliotecas de desenvolvimento de interfaces de usuário mais utilizadas atualmente. A biblioteca de componentes foi criada utilizando a versão mais recente do React, que disponibiliza um novo conceito de lógica de programação chamado *hooks*, possibilitando o reaproveitamento de código. Também foi utilizado o *superset* do *JavaScript*, o *TypeScript*, que adiciona a funcionalidade de tipos na linguagem, possibilitando a criação de códigos com menos erros em *run time*. A biblioteca foi criada com o objetivo de desenvolver aplicativos *web* nativos para uma empresa de desenvolvimento de *software*, pois as outras bibliotecas disponíveis no mercado não possuíam ferramentas necessárias e a possibilidade de utilizar temas para estilização de todos os componentes da biblioteca, além de termos a necessidade de ter total controle dos componentes na criação das aplicações. O resultado obtido foi uma biblioteca com trinta e quatro componentes, que podem ser configurados de maneira global pela funcionalidade do tema, prontos para serem utilizados para a construção aplicações.

Palavras-chave: Biblioteca de componentes; JavaScript; React; TypeScript

ABSTRACT

This work presents the study and development of a components library for React, which is one of the most used libraries for user interface development nowadays. The library was created using the latest version of React, which boasts a new programming logic called hooks, making it easier to reuse code. A superset of JavaScript was also used in this project, the Typescript, making it possible to use types and it is less vulnerable to run-time errors. This library was created with the objective of developing native web applications for a software development company, for other libraries didn't have all the tools and components needed to develop new applications and the possibility of using themes for all components in the library, and also, we needed full control of all components in our developing stages. The library now has thirty-four components, which can be modified globally by the project's theme, making it possible to build apps with it.

Keywords: Component library; JavaScript; React; TypeScript

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APP – *Application*

CSS – *Cascading Style Sheets*

CSS-in-JS – *Cascading Style Sheets in JavaScript*

DOM – *Document Object Model*

ECMA – *European Computer Manufacturers Association*

ES5 – *ECMA SCRIPT 5*

HTML – *Hyper Text Markup Language*

IDE – *Integrated Development Environment*

JS – *Java Script*

JSX – *Java Script XML*

LIB – *Library*

SPA – *Single Page Application*

TS – *Type Script*

WWW – *World Wide Web*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Em 1989, o cientista britânico Tim Berners-Lee desenvolveu a WWW (*World Wide Web*) enquanto trabalhava para a empresa CERN (*European Center for Nuclear Research*) com o objetivo de melhorar o compartilhamento de informação entre cientistas e estudantes de universidades. Logo após, em novembro de 1990, junto com o engenheiro de sistemas Robert Cailliau, Tim criou um documento explicativo mostrando os funcionamentos da WWW, tais como o conceito de navegador, servidores web e a utilização de documentos *hypertext*, sendo eles uma das primeiras versões do atual HTML (*Hyper Text Markup Language*). Dessa forma, em 1991, o projeto foi publicado para fora da empresa CERN e ganhou interesse global. Após esse evento, a internet começou a se expandir se tornando umas das principais ferramentas utilizadas na atualidade com diversas finalidades. (CERN, *A short history of the Web*, 2020)

Com a expansão e popularização da internet, o interesse por desenvolver web sites se tornou cada vez maior. Inicialmente o desenvolvimento era realizado unicamente com HTML básico. Em 1994, com a criação do *JavaScript*, tornou-se possível desenvolver funcionalidades mais avançadas, como melhor estilização da página, abrir janelas dentro do site, entre outras. Porém, a utilização extensiva de *JavaScript* tornava essas páginas pesadas. Para a solução deste problema, a funcionalidade de estilização, que antes era atribuída a essa linguagem, passou a ser atribuída principalmente ao CSS, uma tecnologia que facilita a modificação de estilos de uma página HTML. (SMA *Marketing, The History of Website Design*: 28 *Years of Building the Web*, 2020)

Ainda, com a contínua expansão do interesse por aplicativos *web*, surgiu a necessidade de melhorar ainda mais o poder de seu desenvolvimento. Até então, páginas web que precisassem fazer interações com servidores ou mostrar dados de forma dinâmica realizavam um recarregamento completo da página por interação ou atualização. Porém, isso não era o ideal para o seu desenvolvimento, ainda mais em aplicações web que realizam atualizações diversas vezes por segundo. Por conta desse problema surgiu o conceito de aplicações de página única ou SPA (*Single Page Application*). (Academind, Dynamic vs Static vs SPA, 2019)

Aplicações de página única possuem a funcionalidade de conseguir realizar operações ou atualizações de dados sem a necessidade de recarregar a página por completo, com a utilização do *JavaScript* e com auxílios de bibliotecas. Em SPAs, apenas os pontos que realizaram alguma operação precisam ser recarregados, fazendo com que a experiência seja muito mais otimizada.

Essa característica torna sistemas que realizam muitas operações mais rápido e com experiência de usuário mais otimizada.

Conforme pesquisas realizadas pelo site *StackOverflow* (2020), a biblioteca mais utilizada atualmente para o desenvolvimento de SPAs é o React. React é desenvolvido e mantido atualmente por uma equipe de desenvolvedores do Facebook e por algumas comunidades de desenvolvedores. A biblioteca foi criada por Jordan Walke, um engenheiro de *software* do Facebook, que tinha como objetivo tornar o *feed* de notícias do Facebook dinâmico e com uma boa experiência para o usuário, sem gerar carregamentos completos na página. Em 2011 a biblioteca foi usada no *feed* de notícias do Facebook e em 2012 foi utilizado para a construção do Instagram. No ano de 2013, a biblioteca virou *open source*, tornando seu código e sua utilização abertos ao público. (Wikipedia, *React web framework*, 2020)

Com isso, o React cresceu rapidamente, criando uma comunidade imensa em volta dela, tal como o surgimento de um ecossistema de bibliotecas auxiliares ao React, como bibliotecas que ajudam na estilização, otimização entre outras. Com esses crescimentos, diversas empresas optaram por utilizá-lo como principal ferramenta para desenvolvimento *web*, como o Uber, Netflix e Amazon, tornando a biblioteca confiável atualmente. (*Stackshare*, 2020)

O React tem a funcionalidade de apenas de auxiliar na criação de SPAs, não fornecendo auxílio na estilização de páginas *web*. Para realizar a estilização, ainda é necessário contar com CSS. Então, para desenvolver uma aplicação *web*, todos os estilos ainda devem ser criados utilizando CSS, o que pode se tornar não produtivo, tendo como necessidade de se preocupar com o desenvolvimento tanto da parte de estilos em CSS como a parte de funcionalidades de SPA com o React. Tendo em vista esse desafio, é comum empresas e desenvolvedores recorreram a bibliotecas de componentes para o React, que são responsáveis por disponibilizar componentes para a construção de uma aplicação web já estilizados, o que torna o desenvolvimento muito mais rápido. (Sunscrapers, "*When to use a UI component library in a React project*?", 2020)

A Helios UI, biblioteca de componentes React apresentada nesse trabalho, tem dois objetivos principais. O primeiro é tornar mais ágil, fácil e consistente o desenvolvimento de aplicações *web*. O segundo objetivo é tornar possível a estilização global dos componentes a partir do tema. A biblioteca, por padrão, já possui diversos componentes que podem ser utilizados para o desenvolvimento das mais diversas aplicações, sendo elas simples ou complexas. E, possui componentes para realizar o desenvolvimento de aplicações responsivas, que são aplicações que se

adaptam baseados no tamanho da tela, sendo útil para aplicações que precisem ser executadas em disponíveis mobile, tendo em vista que nos últimos anos dispositivos móveis tem ganhado importância (RIPPON, 2018).

O nome da biblioteca, Helios Ui, tem referência ao sol, que é o centro do sistema solar, sendo ele responsável por manter o sistema todo funcional, fazendo uma analogia que indicaria que a biblioteca proposta nesse trabalho a principal ferramenta para tornar a aplicação do desenvolvedor funcional, como é o sol.

1.1 MOTIVAÇÃO

A Helios UI é proposta para resolver a falta de customização. Muitas bibliotecas disponibilizadas pela comunidade tem o problema de serem difíceis de se adaptarem totalmente a um padrão de estilo já existente de maneira geral, como animações, sombreados, formas, entre outros. A biblioteca proposta nesse trabalho fornece por padrão uma configuração global que afeta diretamente todos os componentes, que é chamada de tema. O tema permite realizar alterações gerações em todos os estilos de todos os componentes da biblioteca. Para configurar o tema basta modificar um único objeto global onde consta as estilizações dos componentes, e então todos os componentes irão se adaptar às regras definidas, tornando a biblioteca extremamente customizável e adaptável para as mais diversas necessidades.

2. MÉTODO DA PESQUISA

Foi necessário um processo de pesquisa de tecnologias necessárias para o desenvolvimento do projeto. Elas foram validadas e escolhidas conforme os objetivos de cada etapa do projeto e por suas vantagens, conforme listadas nos próximos tópicos a seguir. Desta forma pode-se classificar a pesquisa deste trabalho do tipo exploratória qualitativa e de procedimentos pesquisa-ação.

2.1 TYPESCRIPT

> *"TypeScript is an open-source language which builds on JavaScript, one of the world's most used tools, by adding static type definitions.*

> *Types provide a way to describe the shape of an object, providing better documentation, and allowing TypeScript to validate that your code is working correctly." (Microsoft, 2020)*
>
> *TypeScript* é uma linguagem *open source* fundada em cima do *JavaScript*, uma das ferramentas mais utilizadas, adicionando definições estáticas de tipo. Tipos fornecem uma maneira de descrever o formato de um objeto, providenciando uma melhor documentação, e permitindo que o *TypeScript* valide se o código está funcionando corretamente. (Tradução livre)

TypeScript é uma linguagem *open source*, mantida pela Microsoft, construída em cima do *JavaScript*. O *TypeScript* adiciona definições de tipo ao *JavaScript*, tendo em vista que ele não possui essas mesmas definições. Com isso, o desenvolvimento se torna mais seguro, pois é possível determinar em tempo de desenvolvimento se o código está correto ou não, além de tornar o código mais legível e fácil de entender. (*TypeScript*, 2020)

A versão utilizada no projeto proposto nesse trabalho foi a 3.7.5. Essa versão apresenta diversas características para tornar o código mais fácil de ser desenvolvido, como facilidade na verificação de variáveis booleanas, diminuindo a quantidade de código para realizar essas verificações, tipos genéricos, que são tipos que se adaptam à uma variável, entre outras.

2.1.1 Vantagens do TypeScript sobre o JavaScript

A utilização de *TypeScript* no lugar do *JavaScript* gera muitas vantagens. A sua utilização permite a descoberta de problemas durante o período de desenvolvimento do projeto, garantindo que a versão final do código seja menos suscetível a erros (*TypeScript*, 2020), como observado nas **Figura 1 e 2,** demonstra um exemplo escrito em *JavaScript* e mostra um exemplo utilizando *TypeScript*, respectivamente.

Figura 1 - Exemplo comparativo utilizando JavaScript

```
import React from 'react'

const MyTestComponent = (props) => {
  const { text } = props

  return <div>{text.length}</div>
}

// O código abaixo não gera erros na hora do desenvolvimento
// porém no momento de execução resultaria em um erro
const Example = () => {
  return (
    <main>
      <MyTestComponent text={20} />
    </main>
  )
}
```

Fonte: Própria (2020)

Figura 2 - Exemplo comparativo utilizando TypeScript

```
import React from 'react'

type MyTestComponent = {
  text: string
}

const MyTestComponent: React.FC<MyTestComponent> = (props) => {
  const { text } = props

  return <div>{text.length}</div>
}

// O código abaixo gera erros durante o desenvolvimento
// Erro gerado: Type 'number' is not assignable to type 'string'
// Erro traduzido: Uma propriedade do tipo número não pode ser
// atribuida para uma propriedade do tipo string.
const Example = () => {
  return (
    <main>
      <MyTestComponent text={20} />
    </main>
  )
}
```

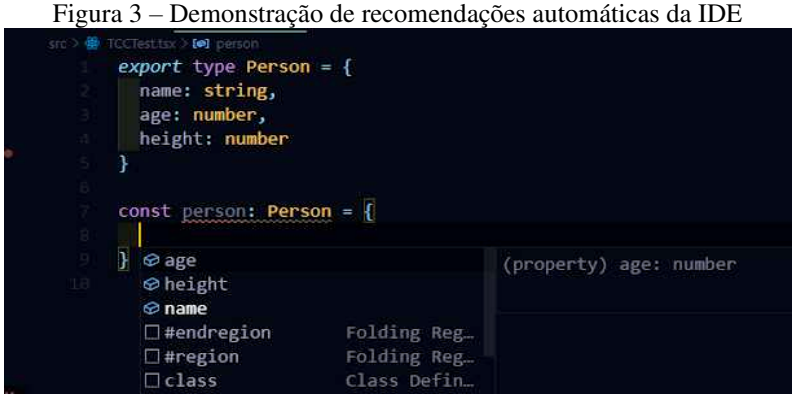
Fonte: Própria (2020)

É possível observar na **Figura 2** que erros (demonstrado como uma linha vermelha estilo onda) são mostrados antes mesmo de compilar a aplicação, contribuindo para um desenvolvimento mais produtivo, eliminando compilações extras desnecessárias, além de ajudar na obtenção de um código mais padronizado.

Assim como citado no site do *Visual Studio Code* (VISUAL STUDIO CODE, 2020), uma outra vantagem para a utilização do *TypeScript* é o melhoramento da ferramenta de inteligência (*intellisense*) da IDE (*Integrated Development Environment*) *Visual Studio Code*®, que permite que ela possua recomendações automáticas de definições de tipo do *TypeScript*, como mostrado na

Figura 3 a seguir. Esta ferramenta permite que o código se torne auto documentado, facilitando a sua utilização e seu entendimento.

Figura 3 – Demonstração de recomendações automáticas da IDE



Fonte: Própria (2020)

Além disso, uma outra vantagem da utilização do *TypeScript* é a sua flexibilidade. Ela permite a criação de códigos reutilizáveis com tipos genéricos e adaptáveis de acordo com a necessidade do projeto. Outra vantagem é a sua extensa comunidade *open source.* (VISUAL STUDIO CODE, 2020).

Logo, após analisar todas estas vantagens, o *TypeScript* se tornou viável para o desenvolvimento do projeto proposto nesse trabalho, tendo em vista que a base de código da biblioteca é extensa, além da necessidade diminuir possíveis erros nos componentes da mesma. (VISUAL STUDIO CODE, 2020).

2.1.2 Funcionamento do TypeScript

Para utilizar o *TypeScript*, é necessário criar um arquivo na raiz do projeto chamado "*tsconfig.json*", como visto na **Figura 4** a seguir.

Figura 4 – Conteúdo do arquivo *tsconfig.json*

```
{
  "compilerOptions": {
    "outDir": "dist",
    "module": "esnext",
    "lib": ["dom", "esnext"],
    "moduleResolution": "node",
    "jsx": "react",
    "sourceMap": true,
    "declaration": true,
    "esModuleInterop": true,
    "noImplicitReturns": false,
    "noImplicitThis": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "suppressImplicitAnyIndexErrors": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "allowSyntheticDefaultImports": true
  },
  "include": ["src"],
  "exclude": ["node_modules", "dist", "example"]
}
```

Fonte: Própria (2020)

Esse arquivo é responsável pela configuração e comportamento do compilador do *TypeScript* em relação ao código. Nesse arquivo são definidos o diretório base da aplicação, o diretório no qual o código deve ser compilado e salvo e para qual versão do *JavaScript* o código será compilado. Todo código *TypeScript* é transformado em um código *JavaScript* no processo de interpretação. Assim, o *TypeScript* apenas beneficia o desenvolvimento do projeto, tendo em vista que o código final é transformado em *JavaScript* comum. (TYPESCRIPT, 2020).

2.2 REACT

> *"React makes it painless to create interactive UIs. Design simple views for each state in your application and React will efficiently update and render just the right components when your data changes. Declarative views make your code more predictable and easier to debug.*" (React, Acesso em 12 de ago. de 2020).
>
> React torna indolor criar interfaces de usuário. Projete visualizações simples para cada estado em sua aplicação e o React vai atualizar eficientemente e renderizar os componentes certos quando os seus dados trocarem. Visualizações declarativas fazem o seu código ser mais previsível e fácil de encontrar problemas. (Tradução livre)

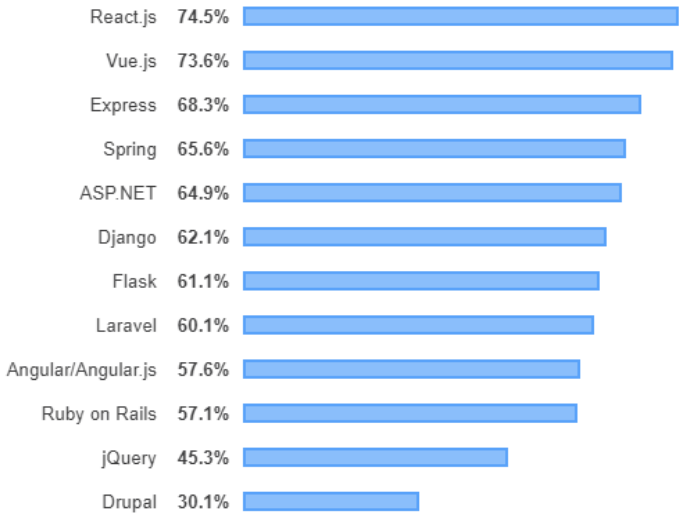
React é uma biblioteca *JavaScript open source* que é utilizada para a criação de interfaces de usuário para uma SPA.

Criado por um engenheiro de software do Facebook chamado Jordan Walke, a biblioteca React até hoje é mantida pelo Facebook e está em constante desenvolvimento até hoje. Atualmente

o React é a biblioteca de *JavaScript* mais famosa e mais utilizada, conforme pesquisa realizada pelo site *StackOverflow* (2019).

A ideia do React é tornar fácil o desenvolvimento de aplicações que possuem constante interação com dados sem a necessidade de recarregar a página como um todo. Além disso, sua utilização é simples, fácil de testar, além de possuir um ecossistema muito grande (REACT, 2020), sendo utilizado pelas maiores empresas de tecnologia atualmente, como mostrado na pesquisa realizada pelo site *StackOverflow* (2019) na **Figura 4**. Essas vantagens se tornaram essenciais para a sua utilização no desenvolvimento da biblioteca de componentes, sendo desta forma o React escolhido como biblioteca JavaScript como padrão deste projeto.

Figura 5 – Gráfico resultado de pesquisa de frameworks mais utilizadas em 2019



Fonte: STACKOVERFLOW. Developer Survey Results 2019, 2019, **StackOverflow**. Disponível em: <https://insights.stackoverflow.com/survey/2019#technology>. Acesso em 19 de set. de 2020.

2.2.1 Funcionamento

O funcionamento do React pode ser comparado com uma árvore. Esta define os elementos HTML de uma página *web*, assim como também o DOM (*Document Object Model*), onde o seu primeiro nó seria uma *tag* HTML (<HTML></HTML>). Desta forma, o React utiliza código *JavaScript* para modificar o DOM e controlar os conteúdos de uma página HTML. (FREE CODE CAMP, 2019).

Cada nó desta árvore é um objeto responsável pelo controle e administração de um elemento HTML, podendo ser chamado também de componente React.

2.2.2 Componente React

Um componente React pode ser definido como um pedaço lógico de código responsável pela descrição e controle do comportamento de um elemento HTML em uma página *web*. Sua estrutura também segue o estilo árvore, podendo ter outros componentes filhos interligados a si. (REACT, 2020).

Uma característica importante do componente para o desenvolvimento em React é a facilidade de reutilização do código já criado em várias partes do mesmo projeto, diminuindo a reescrita de código desnecessárias, tornando-o mais simples e legível, podendo também ser estruturado de duas formas, tanto por classes, quanto por funções. (REACT, 2020).

Os componentes React possuem mecanismos como estados e propriedades (*props*) que são importantes para a estrutura do projeto. Estados são variáveis internas nos quais sempre quando atualizadas, fazem com que o componente se atualize totalmente, processando uma versão com suas propriedades e valores mais atuais. Propriedade é um mecanismo que permite um componente passar dados para outros componentes, seguindo a teoria *one way data flow* do React, que define que dados apenas devem ser transmitidos para componentes filhos e nunca ao contrário. Esse conceito é importante para o maior controle do comportamento de um outro componente, além de criar uma hierarquia no código, facilitando o seu entendimento. (REACT, 2020).

Um exemplo de um componente React simples estruturado em função pode ser observado na **Figura 5**. Sua função é apenas mostrar na página a frase exemplo "*Hello world :)*".

Figura 6 – Exemplo simples de um componente React estruturado em função

Fonte: Própria (2020)

2.2.3 JSX

O JSX é um mecanismo do React que permite que o código *JavaScript* seja escrito de maneira simples, e com uma sintaxe parecida com HTML. No método padrão, é necessário utilizar uma função proveniente do React nomeada *createElement,* que possibilita a criação de um elemento a ser mostrado no DOM, recebendo 3 parâmetros, sendo eles a tag HTML, suas propriedades e o conteúdo do elemento. Porém, esse estilo de criação pode se tornar complexo em projetos muito extensos. Como solução para este problema, o React provê o JSX, que é uma forma mais simples de criar componentes, não dependendo do tamanho da aplicação. Desta forma, todo código JSX é convertido em *JavaScript* após a compilação, fazendo com que o JSX seja apenas uma ferramenta facilitadora no desenvolvimento dos componentes do projeto. (REACT, 2020).

2.3 STYLED-COMPONENTS

Atualmente é possível estilizar (mudar seu estilo, podendo ser mudanças de cor, tamanho, comprimento, sombra, entre outros estilos disponíveis no CSS), de muitas maneiras distintas. É possível utilizar CSS puramente, estilos em formato de objetos *JavaScript*, também como um método chamado CSS-in-JS, que permite utilizar código *JavaScript* para a estilização de um componente React (STYLED COMPONENTS, 2020). A biblioteca de estilos *Styled-Components* (2016) é uma ferramenta para componentes que simplifica o uso do método CSS-in-JS na estilização deles.

2.3.1 Motivação

A biblioteca foi escolhida para estilização pela possibilidade de utilizar propriedades do componente dentro dos estilos, o que permite realizar lógicas dependentes das propriedades. Isso é possível pois a biblioteca cria um componente igual ao existente e adiciona seus estilos a ele. Além disso, a geração dos estilos é feita de forma declarativa, tornando-a organizada e de fácil reutilização. (STYLED COMPONENTS, 2020) Essas vantagens se tornaram viáveis para o projeto final, tendo em vista sua simplicidade e sua praticidade em reutilização de código, principalmente para projetos mais complexos.

2.3.2 Funcionamento

Para a sua utilização, o *Styled-Components* importa uma função da biblioteca de componentes. Essa função recebe como argumento um componente e por meio do método de *string interpolation* (interpolação de texto), a função monta os estilos assim como o CSS, o que é utilizado em HTML padrão.

No momento de execução, internamente, o *Styled-Components* utiliza o componente fornecido como parâmetro e o CSS escrito, gerando uma classe CSS com um nome único, aplicando-a diretamente no componente, fazendo com que ele seja estilizado.

2.3.3 Tema

Um dos fatores mais importantes para a estrutura e funcionalidade do projeto é o tema. O *Styled-Components* tem a funcionalidade de utilizar um objeto global que define padrões de estilo gerais para todo o projeto. Esses padrões podem conter paletas de cores, padrões de espaçamentos, bordas, tamanhos, animações, sombras e comportamentos. (STYLED COMPONENTS, 2020)

Esta funcionalidade fornece os estilos apenas aos componentes filhos. Para isso, é necessário utilizar um componente *Theme Provider* (provedor de tema) no primeiro nível da árvore de componentes do React, que faz com que os estilos criados fiquem disponíveis para todo o projeto sempre quando executado.

Desta forma, sempre quando os estilos são atualizados, todos os componentes filhos que recebem suas propriedades se adaptam automaticamente, sem a necessidade de mudanças manuais em cada um deles. Além disso, esse objeto ajuda na consistência de design em todo o projeto, podendo também facilitar o desenvolvimento de vários outros temas.

2.4 BIBLIOTECA DE COMPONENTES

Biblioteca de componentes é um conceito muito presente no ecossistema do React e nos ecossistemas de desenvolvimento de interfaces de usuário. Esse termo se refere a um conjunto de pedaços de código utilizados para a construção de UIs de maneira mais fácil, sendo possível estilizar de maneira produtiva para alcançar o visual e funcionalidades requeridos.

Uma biblioteca de componentes deve prover naturalmente inúmeros componentes para auxiliar o desenvolver a projetar uma aplicação de forma produtiva. As bibliotecas devem ser de fácil uso, como gerenciar estilos. E deve apresentar uma quantidade de componentes suficientes para atender as mais distintas demandas.

3. DESENVOLVIMENTO

O desenvolvimento do projeto teve como sustentação as quatro fases a seguir:

- Planejamento do projeto: planejamento e criação do *roadmap* para o desenvolvimento do projeto, definindo objetivos a serem cumpridos.
- Ferramentas utilizadas: ferramentas utilizadas para o controle do tempo de conclusão dos objetivos, para o desenvolvimento e publicação do projeto final.
- Configuração do projeto inicial: configuração das tecnologias necessárias no projeto base e estruturação de diretórios.
- Desenvolvimento dos componentes: processo de desenvolvimento da biblioteca de componentes e explicação de uso.

Os tópicos a seguir explicarão com mais detalhes cada objetivo mencionado.

3.1 PLANEJAMENTO DO PROJETO

Os objetivos a serem cumpridos para o desenvolvimento deste projeto foram definidos no segundo semestre do ano de 2019, sendo separados períodos de trabalho para entrega de cada um deles. Cada objetivo foi colocado em uma tabela, como demonstrado na **Tabela 1** a seguir:

Tabela 1 – Definição dos períodos de desenvolvimento do projeto (*roadmap*)

Objetivos (Atividades)	**Produto**	**Período**
Pesquisa, leituras e planejamento da arquitetura do projeto	Estrutura de pastas e tipagens	03/02/2020 – 16/02/2020
Inicialização da criação dos componentes	*Box, Grid, Container, Anchor, Badge, Button, Carousel, Checkbox Input, Collapsible, Divider, Drop, Form, Grid, Heading, Menu*	17/02/2020 – 17/04/2020
Criação de mais componentes	*Number Input, Password Input, Select Input, Table, Text Input, Text Area Input, Text, Toast, Date Input, Masked Input*	18/04/2020 – 03/05/2020
Realização de testes	Site exemplo de visualização dos componentes	04/05/2020 – 10/05/2020
Início da produção da parte escrita	Parte escrita	11/05/2020 – 31/05/2020
Finalização da parte escrita e do projeto	Parte escrita e projeto final	01/06/2020 – 30/06/2020

Além da utilização do *roadmap*, tarefas mais específicas para a criação de cada componente, tais como estruturação e planejamento, foram criadas e adicionadas na ferramenta de organização de tarefas *ClickUp* (2020), que será abordada com mais detalhes no próximo tópico.

3.2 FERRAMENTAS UTILIZADAS

Para este trabalho, foram utilizadas várias ferramentas facilitadoras e essenciais para o progresso do desenvolvimento da biblioteca, tanto para meios de organização, quanto para criação e publicação final do trabalho.

3.2.1 *Visual Studio Code*

Microsoft *Visual Studio Code* (Microsoft 2020) é um editor criado e mantido pela Microsoft, sendo totalmente gratuito pelo fato de ser *open source*. É um editor extremamente customizável por meio de extensões criadas pela comunidade, possui ferramentas muito boas para o desenvolvimento de aplicações, como ferramentas para editar múltiplos arquivos, terminal integrado, navegação de código, entre outros, e também possui baixo custo de processamento para o computador. (VISUAL STUDIO CODE, 2020)

O processo de iniciar a programar com *Visual Studio Code* é extremamente simples, pois ele exige o mínimo de configuração, e possui sua interface amigável e customizável, se tornando um ótimo ambiente para se programar.

3.2.2 Click Up

Para organizar projetos extensos, como a biblioteca de componentes proposta nesse trabalho, é importante ter uma ferramenta que ajude a montar uma sequência de tarefas a serem executadas, para se ter total controle de tempo de execução e histórico das tarefas. A ferramenta escolhida para atender esse objeto foi o *ClickUp (*2020*)*. Essa ferramenta possibilita, de maneira simples e completa, montar um plano de atividades contabilizando datas, dificuldade e objetivos de uma tarefa.

É possível também definir nessa ferramenta métodos de classificação de complexidade e tempo de cada tarefa. Neste projeto, utilizamos o método Fibonacci (complexidade da tarefa x tempo de execução) para a pontuação de cada uma delas, utilizando sua sequência de números (1, 2, 3, 5, 8, 13...). Baseado nessas pontuações atribuídas é possível estimar o tempo de execução de cada objetivo.

Dessa forma, a cada período de 5 dias úteis era definido um conjunto de tarefas pontuadas a serem executadas, variando entre 25 a 32 pontos no total por ciclo. Todo esse processo de desenvolvimento foi possível por conta das funcionalidades disponíveis pelo *ClickUp*.

3.2.3 GitHub

GitHub é um serviço gratuito utilizado por milhões de desenvolvedores para hospedar, de forma gratuita, uma base de código. Por padrão, todo repositório de código adicionado no GitHub pode ser acessado e consultado por qualquer pessoa, se tornando uma ferramenta muito útil para o seu desenvolvimento e compartilhamento com a comunidade open-source. (GITHUB, 2020)

Esse serviço ajuda no versionamento de código que garante que mudanças no projeto principal possam ser feitos de maneira segura e consistente, evitando problemas como duplicação ou perda de código, que são problemas comuns quando se trabalha projetos de grande escala. O serviço também permite que vários desenvolvedores do mundo todo colaborem em projetos que achem interessantes, com o sistema de *commits*, que são requisições para alteração de código no projeto principal, que precisam ser aceitos pelo representante do projeto. (GITHUB, 2020)

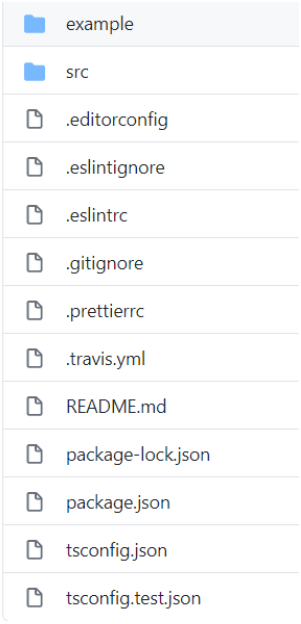
3.3 CONFIGURAÇÃO INICIAL DO PROJETO

Para que a biblioteca possa ser utilizada é necessário definir algumas configurações como uma estrutura de diretórios intuitiva e de fácil de leitura tal como definir quais ferramentas utilizar para tornar possível utilizar o React e o *TypeScript*.

3.3.1 Estrutura de diretórios

A estrutura de diretórios foi projetada para a facilitar a localização dos componentes individualmente. A raiz do projeto é demonstrada na **Figura 7**.

Figura 7 – Estrutura de diretórios do projeto



Fonte: Própria (2020)

O primeiro diretório do projeto é chamado de *"example"*. Ele é responsável por armazenar uma aplicação React de testes, onde os componentes são renderizados e testados.

Os arquivos *package.json* e *package-lock.json* são responsáveis pela configuração da versão da biblioteca e dos pacotes do projeto.

Os arquivos *.eslintrc, .eslintignore, .prettierrc, travis.yml* e *.editorconfig* são responsáveis por controlar padrões de estilo de código, como por exemplo se é permitido ou não usar ponto e vírgula. Essas regras de estilo de código são puramente para manter um padrão e não surtem efeito nas funcionalidades do projeto e são utilizadas apenas pelo editor.

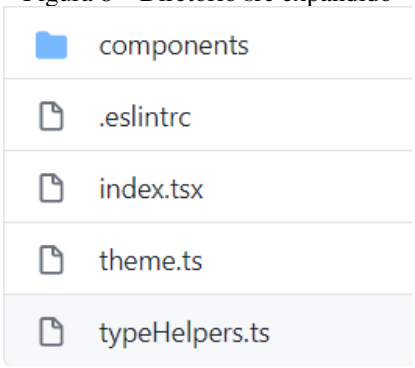
O arquivo *.gitignore* é responsável por configurar o Git para definir quais arquivos não vão ser enviados para o repositório no GitHub.

O arquivo *README.md* tem a finalidades de apresentar de maneira geral e simples o objetivo da biblioteca. Esse arquivo é mostrado como página inicial no *GitHub*.

Os arquivos *tsconfig.json* e *tsconfig.test.json* tem como objetivo configurar o compilador *TypeScript*. Sem esses arquivos de configuração a linguagem não funciona, deixando de trazer seus benefícios.

O diretório *src* é o principal diretório do projeto. Nele estão contidos todos os arquivos e códigos principais, além de todos os componentes desenvolvidos na biblioteca. A pasta expandida é mostrada na **Figura 8.**

Figura 8 – Diretório src expandido

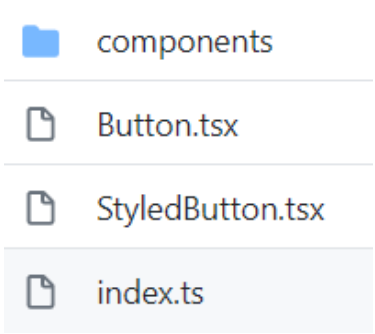


Fonte: Própria (2020)

Dentro do diretório *src*, existe o diretório *components*, o arquivo *index.tsx, theme.ts*, *typeHelpers.ts* e *.eslintrc*. A finalidade do *index.tsx* é exportar todo os componentes e funcionalidades do tema para poderem serem utilizados, sem esse arquivo é impossível utilizar os componentes. O arquivo *theme.ts* contém o tema padrão junto com algumas definições de tipagens do tema. O arquivo *typehelpers.ts* contém tipagens genéricas para facilitar o desenvolvimento dos componentes. Por fim, o arquivo *components* contém todos os componentes da biblioteca.

Um componente deve estar dentro de sua própria pasta com o nome do componente. Dentro dela, é necessário também ter um arquivo *tsx* com o mesmo nome do diretório criado, onde estará contido o código principal do componente. Nela também deve existir um arquivo *index.ts* para exportar o código do componente para ser utilizado por outras pastas. É possível que um componente tenha uma pasta *components*, tendo em vista que um componente pode ser constituído de outros componentes, além da possibilidade de ele ter mais arquivos para o auxílio na sua funcionalidade e nos estilos. Um exemplo disso é mostrado na **Figura 9**.

Figura 9 – Exemplo de diretório de um componente da biblioteca



Fonte: Própria (2020)

3.3.2 Create-react-library

Create-react-library é uma ferramenta open source que facilita a configuração inicial de um projeto de biblioteca React. Ela inicialmente cria um diretório com React, *ESLint* e *TypeScript* pré configurado e com uma pasta example que é um site em React vazio para poder demonstrar os componentes da biblioteca em prática. (NPM, 2020)

Para utilizar a ferramenta é necessário abrir o terminal e rodar o comando *create-react-lib* e seguir o passo a passo mostrado no terminal e então no final de tudo a configuração inicial vai ser criada.

3.4 DESENVOLVIMENTO DOS COMPONENTES

Para desenvolver um componente, é necessário quebrar em etapas para garantir que o produto terá o resultado desejado de maneira organizada e consistente. O desenvolvimento é dividido em 5 etapas:

- Planejamento do conceito inicial
- Prototipação
- Desenvolvimento
- Realização de testes
- Envio do código final ao GitHub.

3.4.1 Conceito inicial

É primeira fase do processo. Nela é discutido quais são as funcionalidades e qual a finalidade do componente tal como qual seria o impacto de adicionar esse componente no código principal da biblioteca. Durante essa fase é criado uma tarefa no *ClickUp* que fica reservada para uma futura onda de execução. Nessa fase também é feito por escrito, de maneira simples, os comportamentos que o componente deve possuir

3.4.2 Prototipação

Prototipação é fase em que ocorre o desenvolvimento de um exemplo simples do componente. E então é realizado alguns testes simples e baseado nisso é definido como deve ser desenvolvido as funcionalidades finais, tais como o peso da tarefa de desenvolvimento, uma sequência de passos a serem seguidos e também é feito uma demonstração visual de como o componente deve se parecer em sua versão final

3.4.3 – Desenvolvimento dos componentes

Para que a biblioteca seja viável no sentido de ser produtiva e resolver a maioria dos problemas, é necessário que ela disponibilize uma grande quantidade de componentes. Foram desenvolvidos trinta e quatro componentes que auxiliam desde a estrutura da página até iterações com usuário como formulários e eventos de resposta.

3.4.4 Desenvolvimento do código

A fase de desenvolvimento do código é o momento de desenvolvimento final do componente. Nessa fase é realizado toda a parte de desenvolvimento de código necessária para o funcionamento da versão final do componente, como configuração pela funcionalidade do tema, estilos visuais e mecânicas de posicionamento e espaçamento.

3.4.5 Testes

A fase de testes tem a finalidade de garantir que o código desenvolvido na fase anterior esteja funcionando corretamente de maneira que atenda a concepção criada na primeira fase. Caso os testes tenham resultados positivos o componente é liberado para a quinta e última fase, caso contrário o componente volta para a fase anterior e então passa por uma nova onda de testes. Os testes geralmente consistem em verificar se as funcionalidades e visuais estão como desejados e se não apresenta nenhum tipo de erro ou alerta no console.

3.4.6 Upload da biblioteca no GitHub

Nessa última fase, através da ferramenta *GitHub*, é feito o envio do código atualizado para o repositório de código principal. Quando o envio é feito, a biblioteca fica em uma versão mais recente e o componente adiciona fica disponível para utilização para o desenvolvimento de projetos.

4. UTILIZAÇÃO DA BIBLIOTECA

Para a utilização da biblioteca, é necessário configurar o projeto React para que os componentes e o tema estejam disponíveis para o desenvolvimento de interfaces. Este tópico exemplifica a sua configuração e estilização (caso desejada).

4.1 INSTALAÇÃO DA BIBLIOTECA EM UM NOVO PROJETO REACT

Para a instalação da biblioteca de componentes Helios UI em um novo projeto React, é necessário acessar o diretório raiz do projeto e executar, via linha de comando, o comando:

npm install --save styled-components helios-ui

Dessa forma, a biblioteca estará instalada nos módulos do projeto, como pode ser visto no diretório *node_modules* do projeto.

4.2 CONFIGURAÇÃO DA BIBLIOTECA EM UM NOVO PROJETO REACT

Com a biblioteca instalada conforme demonstrado no tópico anterior, é ainda necessário configurar o projeto para a utilização da biblioteca. No topo da árvore da aplicação, deve-se adicionar o componente Helios da biblioteca, onde o mesmo ainda deve receber como propriedade o tema a ser utilizado na aplicação, como visto na Figura 10 abaixo. Dessa forma, o tema estará disponível para toda a árvore de componentes da aplicação por meio do *context* API do React, que permite passar uma propriedade a todos os seus componentes filhos.

Figura 10 – Configuração inicial da biblioteca em um novo projeto React

```
import React from 'react'
import ReactDOM from 'react-dom'
import { App } from './App'
import { Helios } from 'helios-ui'
import { theme } from './utils/theme'

ReactDOM.render(
  <Helios theme={theme}>
    <App />
  </Helios>,
  document.getElementById('root')
)
```

Fonte: Própria

4.3 EXEMPLO DE APLICAÇÃO UTILIZANDO A BIBLIOTECA

Como primeiro passo, é preciso importar os componentes da biblioteca necessários para a construção da interface da aplicação. Dessa forma, é possível montar a interface para a calculadora, utilizando as propriedades dos componentes para a estilização da aplicação, conforme visto na **Figura 11**.

Figura 11 – Código utilizando componentes da biblioteca

```
import React from 'react'
import { Box, NumberInput, Button, Text } from 'helios-ui'

export const App: React.FC = () => {
  return (
    <Box style={{ width: 600 }} direction='column' padding='large'>
      <Box justifyContent='space-between'>
        <NumberInput label='Valor A' />
        <NumberInput label='Valor B' />
      </Box>
      <Box justifyContent='space-between'>
        <Button>Sum (+)</Button>
        <Button>Subtract (-)</Button>
        <Button>Multiply (*)</Button>
        <Button outline>Divide (/)</Button>
      </Box>
      <Box justifyContent='center' margin='large'>
        <Text>Result:</Text>
      </Box>
    </Box>
  )
}
```

Fonte: Própria

Assim, será necessário criar as funções básicas para as operações da calculadora. Para isso, foi importada a *hook useState*, que retorna uma variável e uma função que atualiza essa variável. Essas variáveis foram utilizadas para atualizar os dois valores da calculadora e seu resultado pós operações. Conforme a **Figura 12**, é possível observar as operações aplicadas.

Figura 12 – Operações para a aplicação de calculadora utilizando *useState*

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

export const App: React.FC = () => {
  const [valueA, setValueA] = useState(0)
  const [valueB, setValueB] = useState(0)
  const [result, setResult] = useState(0)

  const onValueAChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setValueA(+event.target.value)
  }

  const onValueBChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setValueB(+event.target.value)
  }

  const calculate = (operation: string) => () => {
    switch (operation) {
      case 'sum':
        setResult(valueA + valueB)
        return
      case 'sub':
        setResult(valueA - valueB)
        return
      case 'multiply':
        setResult(valueA * valueB)
        return
      case 'divide':
        setResult(valueA / valueB)
        return
      default:
        return
    }
  }

  return (/* JSX Value Here */)
}
```

Fonte: Própria

Com isso, é necessário apenas integrar as operações e funções da calculadora com os componentes da biblioteca, através das propriedades dos mesmos, conforme **Figura 13** a seguir.

Figura 13 – Funções e operações matemáticas integradas com os componentes da Biblioteca

```
import React, { useState } from 'react'
import { Box, NumberInput, Button, Text } from 'helios-ui'

export const App: React.FC = () => {
  /*
    Logical Functions here
  */
  return (
    <Box style={{ width: 600 }} direction='column' padding='large'>
      <Box justifyContent='space-between'>
        <NumberInput onChange={onValueAChange} label='Valor A' />
        <NumberInput onChange={onValueBChange} label='Valor B' />
      </Box>
      <Box justifyContent='space-between'>
        <Button onClick={calculate('sum')}>Sum (+)</Button>
        <Button outline onClick={calculate('sub')}>Subtract (-)</Button>
        <Button onClick={calculate('multiply')}>Multiply (*)</Button>
        <Button onClick={calculate('divide')} outline>Divide (/)</Button>
      </Box>
      <Box justifyContent='center' margin='large'>
        <Text>Result: <b>{result}</b></Text>
      </Box>
    </Box>
  )
}
```

Fonte: Própria

É possível observar que as funções matemáticas utilizaram os valores dos estados da aplicação, onde suas atualizações foram implementadas nos componentes de *NumberInput* e chamadas quando os *Buttons* são pressionados, cada um passando como argumento sua respectiva operação matemática.

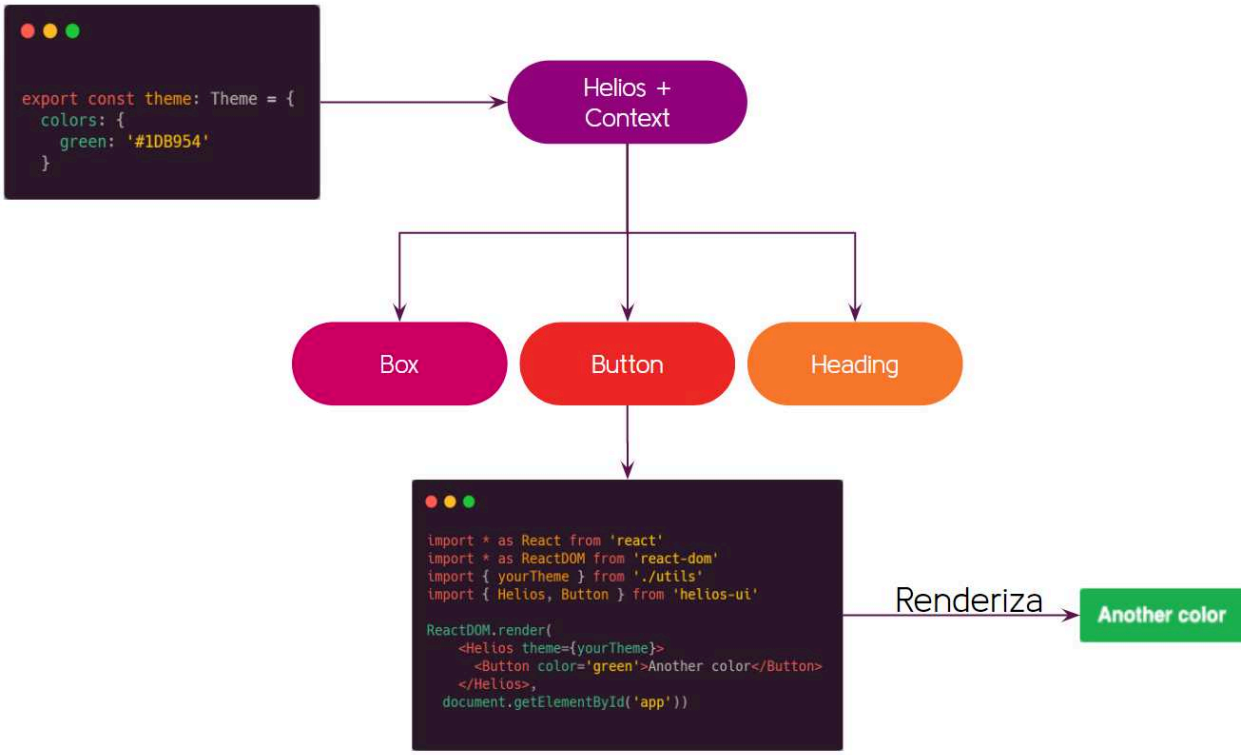
4.4 EDITANDO O TEMA

É possível mudar as propriedades do tema da biblioteca conforme necessário, sendo possível criar temas próprios seguindo as definições de tipo da biblioteca. Para isso, qualquer valor do objeto global que for alterado é automaticamente atualizado em todos os componentes da biblioteca. A **Figura 14** demonstra o funcionamento do tema da biblioteca, onde um valor de cor *green* é definida. Dessa forma, essa cor é atualizada em todos os componentes a partir do *Context* API do React, que fornece esses dados a seus componentes filhos.

Para facilitar a edição do tema, basta importar a definição de tipo *Theme,* que é exportada pela biblioteca, atribuir esse tipo a uma variável e seguir as recomendações propostas pela

funcionalidade de recomendação de código, tendo em vista que o editor utilizado tenha suporte para esta funcionalidade.

Figura 14 – Funcionamento da biblioteca utilizando o objeto global de tema



Fonte: Própria

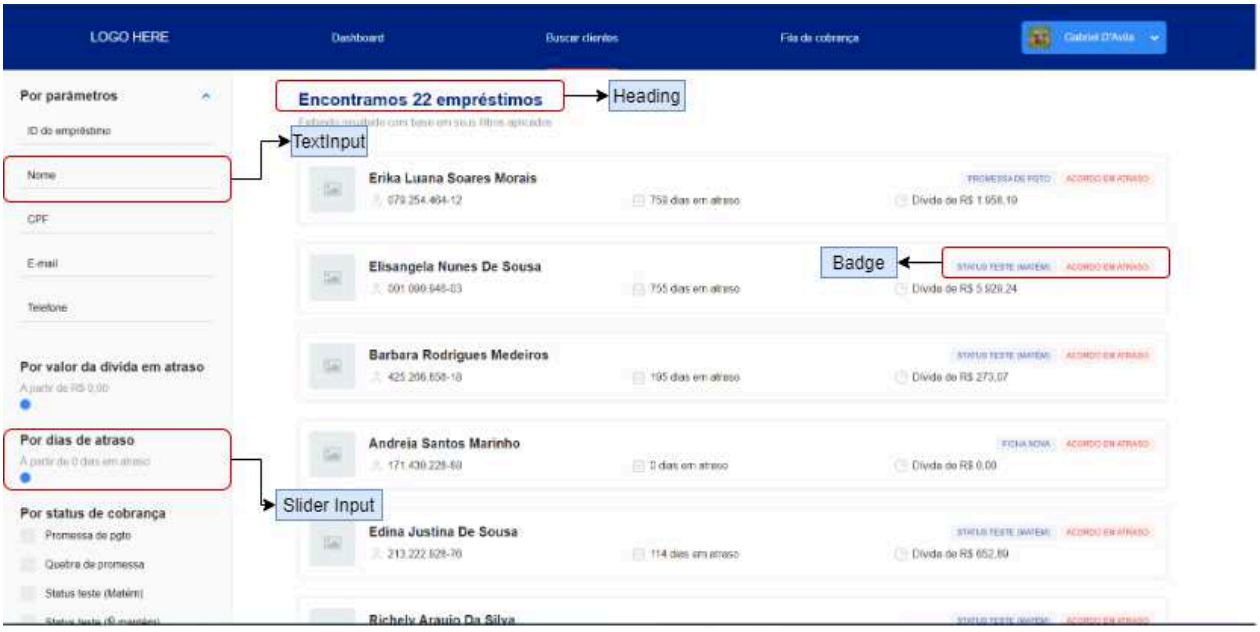
5. RESULTADOS

Com este projeto é possível discutir o desenvolvimento de uma biblioteca de componentes altamente customizável para a construção de aplicações web que utilizam React. É possível observar na **Figura 15** e na **Figura 16** exemplos de sua utilização**.** A lista de todos os componentes criados é vista no **Apêndice A.**

Figura 15 – Exemplo de código em *JavaScript* do componente, do seu código em HTML gerado e da sua renderização final

Fonte: Própria (2020)

Figura 16 - Exemplo de uma aplicação real utilizando os componentes da HeliosUI



Fonte: Própria (2020)

6. CONCLUSÕES

A Helios UI como uma biblioteca de componentes torna mais produtivo o desenvolvimento de aplicações web e altamente customizável e adaptável por conta da funcionalidade de estilos globais, o tema, o que reafirma que os objetivos desse trabalho foram executados com sucesso.

Tendo em vista que todo o projeto foi desenvolvido com *TypeScript*, a produtividade da utilização dos componentes se torna bem segura, por conta da validação de tipos em tempo de desenvolvimento, os componentes em geral se tornaram menos suscetível a erros. Vale adicionar que as recomendações de código geradas pelo *TypeScript* em IDEs que possuem suporte a essa linguagem também facilita o desenvolvimento, aumentando a produtividade.

7. REFERÊNCIAS

FERNANDES, Diego. TypeScript: Vantagens, mitos, dicas e conceitos fundamentais. **RocketSeat,** 2019**.** Disponível em: <https://blog.rocketseat.com.br/typescript-vantagens-mitos-conceitos/>. Acesso em: 23 de mar. de 2020.

RIPPON, Carl. Why TypeScript with React ?. **Carl Rippon,** 2018**.** Disponível em: <https://www.carlrippon.com/why-typescript-with-react/>. Acesso em: 23 de mar. de 2020.

KASUNDRA, Prayaag. Why and Where Should you Use React for Web Development ?. **Simform,** 2020. Disponível em: <https://www.simform.com/why-use-react/>. Acesso em: 23 de mar. de 2020.

PANDIT, Nitin. What and Why React.js. **C Sharp Corner,** 2020. Disponível em: < />. Acesso em: 18 de fev. de 2020.

MOHAN, Mehul. How React Works under the hood. **Free Code Camp,** 2019. Disponível em: <https://www.freecodecamp.org/news/react-under-the-hood/>. Acesso em: 23 de mar. de 2020.

A short history of the Web. **CERN,** 2020. Disponível em: <https://home.cern/science/computing/birth-web/short-history-web>. Acesso em 23 de mar. de 2020.

SHELLEY, Ryan. The History of Website Design: 28 years of Building the web. **SMA Marketing,** 2019 Disponível em: <https://www.smamarketing.net/blog/the-history-of-website-design>. Acesso em: 23 de mar. de 2020.

React (web framework). **Wikipedia,** 2020. Disponível em: <https://en.wikipedia.org/wiki/React_(web_framework)>. Acesso em: 23 de mar. de 2020.

JavaScript. **Wikipedia**, 2020. Disponível em: <https://en.wikipedia.org/wiki/JavaScript>. Acesso em: 23 de mar. de 2020.

Choose Between Traditional Web Apps and Single Page Apps (SPAs). **Microsoft,** 2020**.** Disponível em: <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>. Acesso em 23 de mar. de 2020.

SCHWARZMÜLLER, Maximilian. Dynamic Websites VS Static Pages VS Single Page Apps. **Academind,** 2019. Disponível em: <https://academind.com/learn/web-dev/dynamic-vs-static-vs-spa/>. Acesso em 20 de ago. 2020.

VETTERNEO, Natalia. When to use a UI componente library in a React project?. **Sunscrapers**, 2020. Disponível em: <https://sunscrapers.com/blog/when-to-use-a-ui-component-library-in-a-react-project/>. Acesso em 18 de ago. de 2020.

React. **Stackshare,** 2018. Disponível em: <https://stackshare.io/react>. Acesso em 16 de abr. de 2020.

STYLED COMPONENTS. Disponível em: <https://styled-components.com/>. Acesso em: 23 de mar. de 2020

REACT. Disponível em: <https://reactjs.org/>. Acesso em: 23 de mar. de 2020.

TYPESCRIPT. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 23 de mar. de 2020.

JAVASCRIPT. Disponível em: <https://www.javascript.com/>. Acesso em 23 de mar. de 2020.

VISUAL STUDIO CODE. Disponível em: <https://code.visualstudio.com/>. Acesso em 23 de mar. de 2020.

CLICK UP. Disponível em: <https://clickup.com/>. Acesso em: 23 de mar. de 2020.

GIT HUB. Disponível em: <https://github.com/>. Acesso em 23 de mar. de 2020.

VISUAL STUDIO CODE. TypeScript in Visual Studio Code. **Visual Studio Code**, 2020. Disponível em: <https://code.visualstudio.com/docs/languages/typescript>, Acesso em: 20 de out de 2020**.**

NPM. Dísponível: <https://www.npmjs.com/>. Acesso em: 22 de fev de 2020**.**

APÊNDICE A

Neste apêndice encontram-se todos os componentes desenvolvidos da **Helios UI** em ordem alfabética, com exemplificações e/ou ilustrações. Ao final é explicado o componente essencial da biblioteca, o componente *Helios*.

Anchor

Anchor é responsável por realizar navegações para outras páginas web ou para realizar navegações para páginas internas da aplicação. Esse componente recebe como propriedades um endereço para realizar uma navegação tal como propriedades para estilização, como cores.

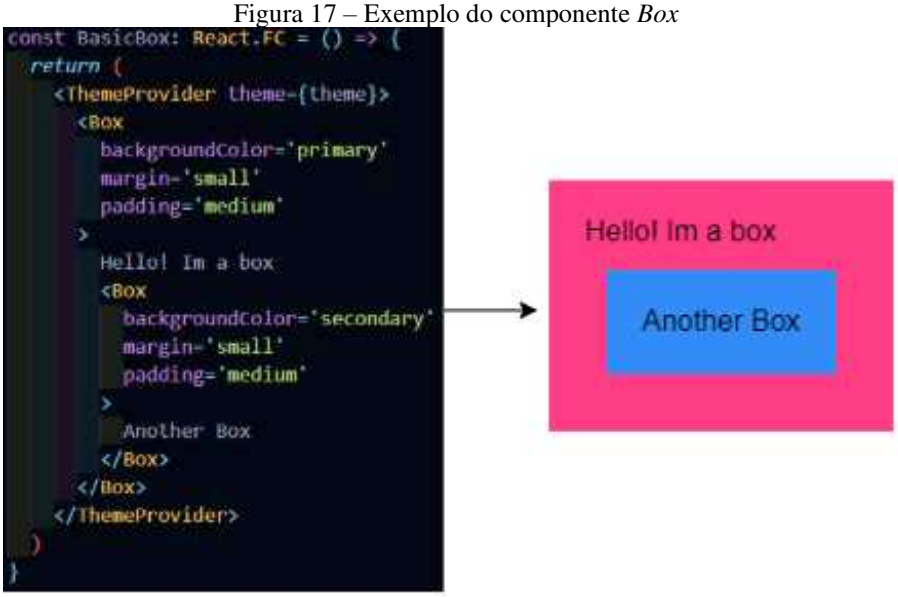
Badge

Badge é utilizado geralmente em para dar uma informação para o usuário após a realização de algum evento na aplicação como um clique de botão ou um envio de formulário. A mensagem transmitida através do *badge* tem a finalidade de guiar o usuário e gerar confirmações, como por exemplo se aconteceu algum erro inesperado.

Box

Box é componente base da biblioteca. Ele é responsável por ser um envoltório de outros componentes para auxilia com posicionamentos, espaçamentos e tamanho. Alguns componentes da biblioteca dependerem diretamente desse componente. Portanto, ele pode ser utilizando tanto para a estrutura de uma página quanto para o desenvolvimento de um componente.

Figura 17 – Exemplo do componente *Box*



Fonte: Própria (2020)

Button

Button é um componente visual que possui a finalidade de realizar alguma funcionalidade quando o usuário da aplicação desenvolvida interage com um clique. Por padrão, esse componente já possui animações que reagem a partir de eventos do mouse do usuário e possui propriedades para trocar cores, tamanho e eventos de usuário, tornando-o altamente personalizável.

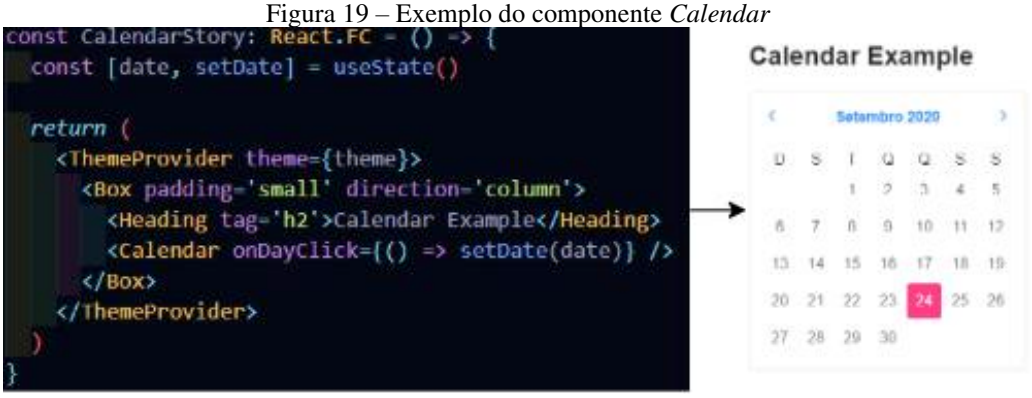
Figura 18 – Exemplo do componente *Button*

Fonte: Própria (2020)

Calendar

Calendar é um componente complexo que depende de outros componentes como Box e Button. Ele tem a finalidade de disponibilizar uma visualização de calendário, com dias, anos e semanas. É possível realizar entre anos para visualizar datas futuras e passadas.

Figura 19 – Exemplo do componente *Calendar*

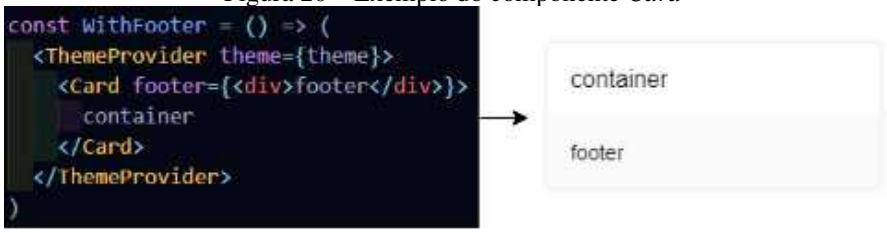


Fonte: Própria (2020)

Card

O componente *Card* tem o objetivo de criar visualizações com um efeito de elevação através da aplicação de sombreados. Esse sombreado pode ser customizado no tema para poder atender a qualquer necessidade. Geralmente esse componente é utilizado para criar componentes mais chamativos do que os demais.

Figura 20 – Exemplo do componente *Card*



Fonte: Própria (2020)

CheckBox

CheckBox é um componente de input responsável por marcar afirmações ou informações como verdadeiro ou falso. É constituído de um texto com uma caixa na lateral que possui um ícone para indicar se está marcado ou não. Todas as funcionalidades que controlam o input são customizáveis. Um caso de uso seria aceitar os termos de um site na hora de realizar um registro.

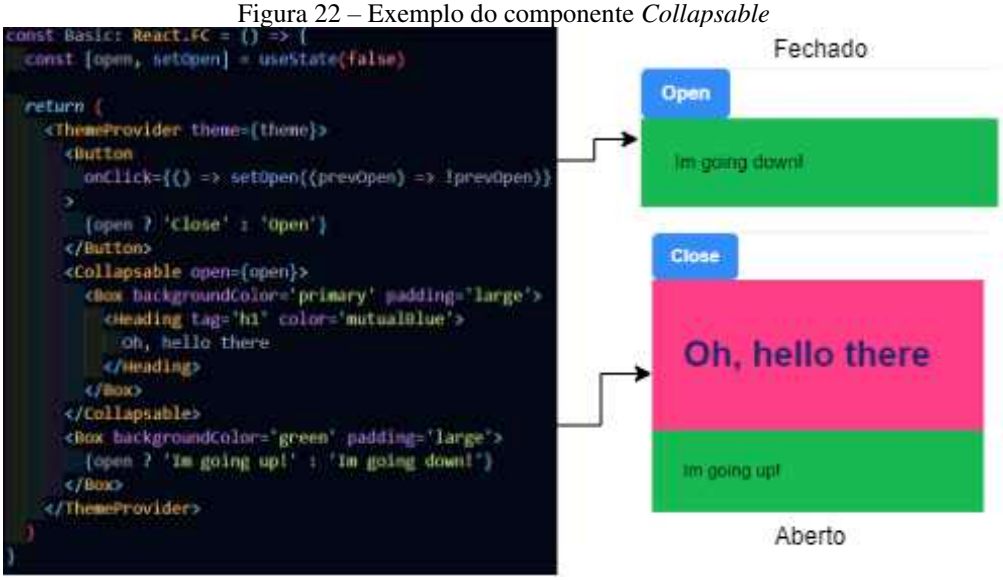
Figura 21 – Exemplo do componente *CheckBox*

Fonte: Própria (2020)

Collapsable

Collapsable é um componente especial que server como envoltório de componentes que desejam ter a funcionalidade de aparecer e desaparecer baseado em uma ação do usuário. Esse componente recebe como propriedade uma variável que define se está visível ou não e recebe uma função que troca o valor dessa variável.

Figura 22 – Exemplo do componente *Collapsable*

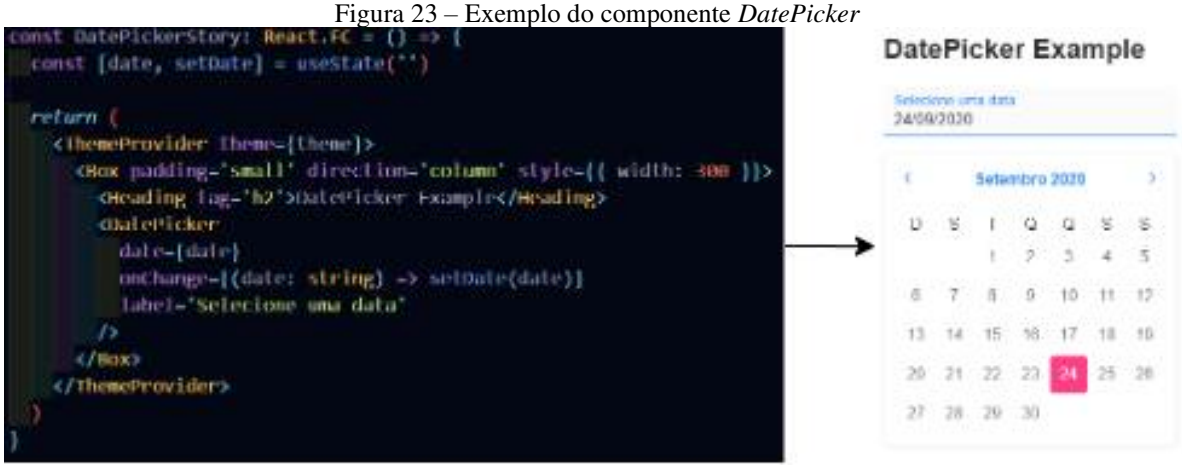


Fonte: Própria (2020)

Date Picker

É um componente construído a partir do *Text Input* e do *Calendar*. Ele tem a finalidade de criar um input para selecionar datas e períodos, geralmente sendo utilizado em formulários. Todos os eventos e funcionalidades são customizáveis para facilitar a utilização.

Figura 23 – Exemplo do componente *DatePicker*



Fonte: Própria (2020)

Divider

O *Divider* é um componente semelhante a tag <hr/> do HTML. O objetivo do *Divider* é separar contextos para melhorara a visualização. Ele geralmente é utilizado para separar componentes com funcionalidades ou contextos distintos que estão na mesma tela ou visualização.

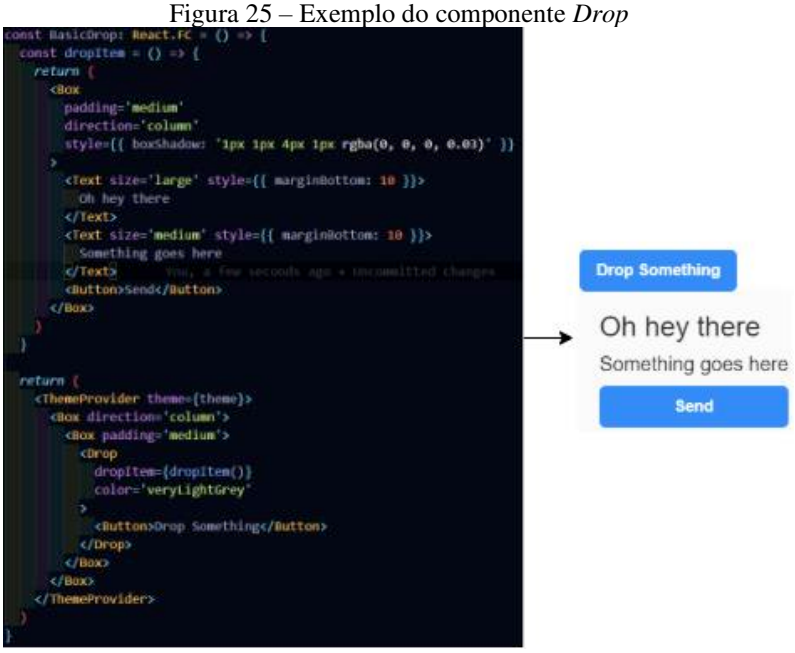
Figura 24 – Exemplo do componente *Divider*

Fonte: Própria (2020)

Drop

É um componente especial que adiciona uma caixa flutuante embaixo de um componente que aparece e desaparece baseado em ações do usuário. É um componente útil para a construção de menus, mensagens de ajuda e *inputs*.

Figura 25 – Exemplo do componente *Drop*



Fonte: Própria (2020)

Grid

Grid é responsável por resolver problemas estruturais mais complexos. Geralmente é utilizado para montar leiautes de maneira responsiva, ou seja, montar leiautes que se adaptam baseados no tamanho da tela, sendo útil quando existe a necessidade de construir uma aplicação que funcione bem em dispositivos moveis.

Grid Item

É um componente para ser utilizado exclusivamente com o componente *Grid*. O *Grid* Item representa um elemento no *Grid* conforme as especificações de regra de estilos do CSS. Ou seja, um *Grid Item* é um envoltório para outros componentes que devem fazer parte da *Grid* para construir uma estrutura complexa.

Image

O componente *Image* tem a finalidade de mostrar imagens na tela, ele tem sua funcionalidade parecida com a funcionalidade da tag *img* do HTML normal, porém com algumas adições de estilo e funcionalidade para facilitar o estilo.

Figura 26 – Exemplo do componente I*mage*

Fonte: Própria (2020)

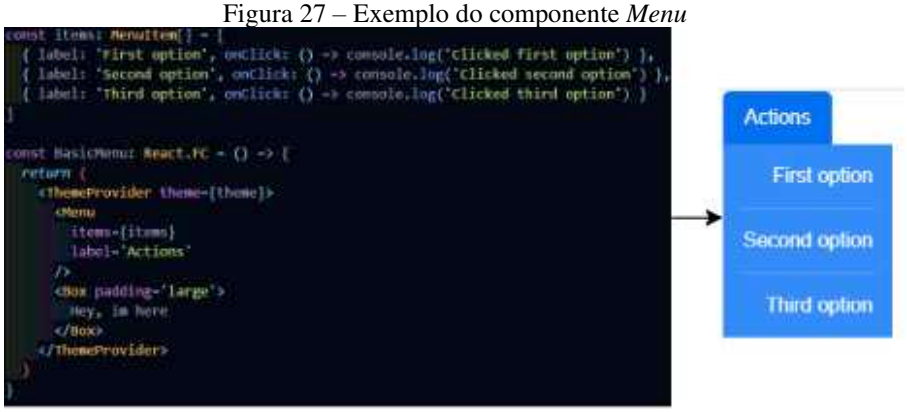
Input

Input é o componente base para a construção de todos os inputs da biblioteca. Ele possui as funcionalidades básicas tais como todos os estilos, que estão ligados diretamente ao tema. Também possui animações para eventos de interações com o usuário.

Menu

Componente construído a partir do *Drop* e do *Button*. Ele é um botão que, a partir de um clique do usuário, fornece uma lista de botões para realizar ações. Todas as interações com esse componente são realizadas com animações para tornar a usabilidade mais agradável. Todas as funcionalidades desse componente são customizáveis por meio do sistema de propriedades do React.

Figura 27 – Exemplo do componente *Menu*

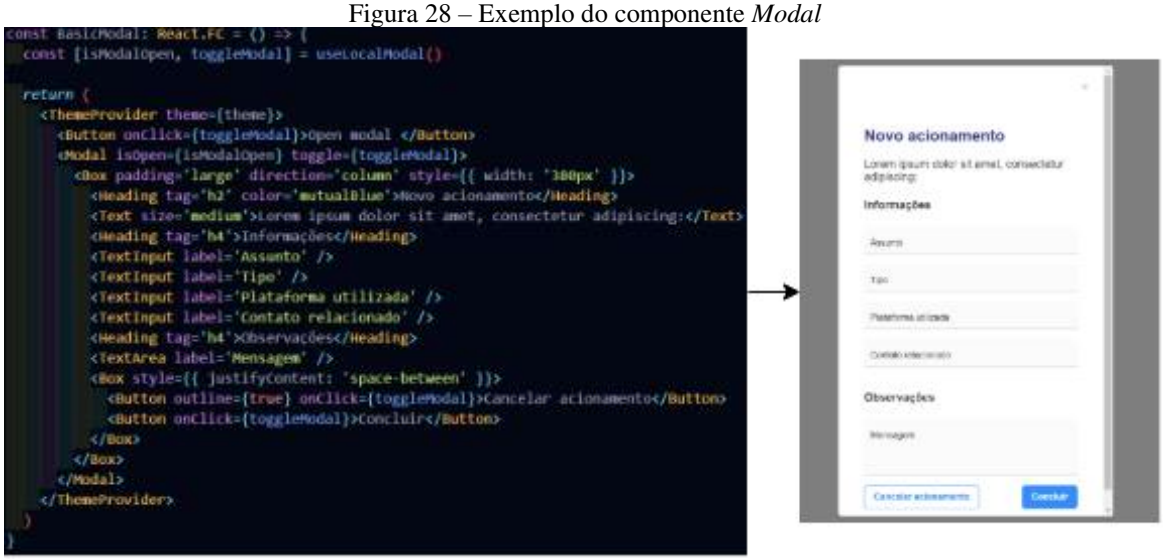


Fonte: Própria (2020)

Modal

O *Modal* é uma janela que abre em cima da aplicação toda, ofuscando os demais componente através de um fundo preto meio opaco. Esse componente tem a finalidade de realizar ações importantes ou até mesmo mostrar informações chamativas. Esse componente deve receber uma propriedade que define quando ele está visível ou não.

Figura 28 – Exemplo do componente *Modal*



Fonte: Própria (2020)

NumberInput

É um *input* com uma construção parecida com o *TextInput*, porém, a diferença é que ele aceita apenas dados numéricos como valor. Assim como o *TextInput*, todas as suas funcionalidades são customizáveis.

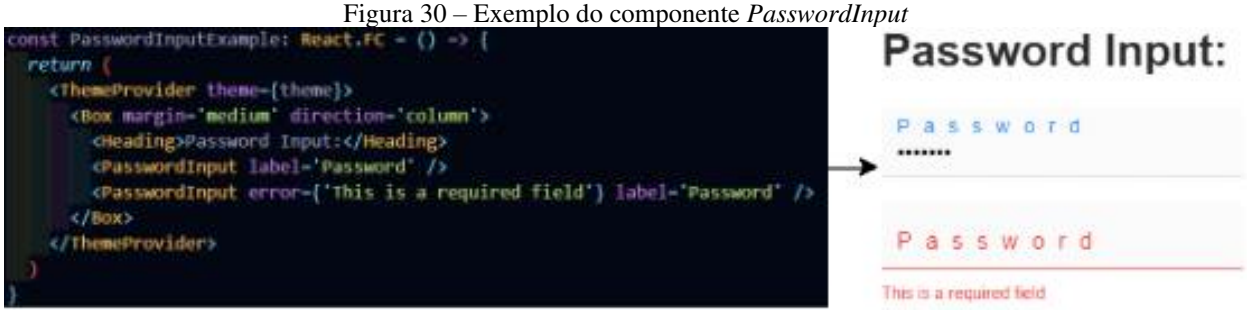
Figura 29 – Exemplo do componente *NumberInput*

Fonte: Própria (2020)

Password Input

Assim como o *TextInput* e o *NumberInput*, esse componente foi construído a partir do componente base Input. O *PassworInput* é utilizado para receber uma senha, então não é possível ler os conteúdos escritos. Geralmente é utilizado para realizar processo de autenticação em aplicações.

Figura 30 – Exemplo do componente *PasswordInput*



Fonte: Própria (2020)

ProgressBar

ProgressBar é um componente utilizado para realizar a visualização de progresso. Geralmente é utilizado para medir o progresso de operações longas, como requisições pesadas para um servidor. Ela recebe um número como propriedade que varia de 0 a 100, que define o progresso.

Select

É um *input* responsável por apresentar uma lista de opções para que o usuário possa escolher uma. A caixa de opções aparece embaixo da caixa de texto. Os valores possíveis de serem selecionados são do tipo texto. Esse *input*, assim como os demais citados nesse trabalho, é construído em cima do componente *Input*.

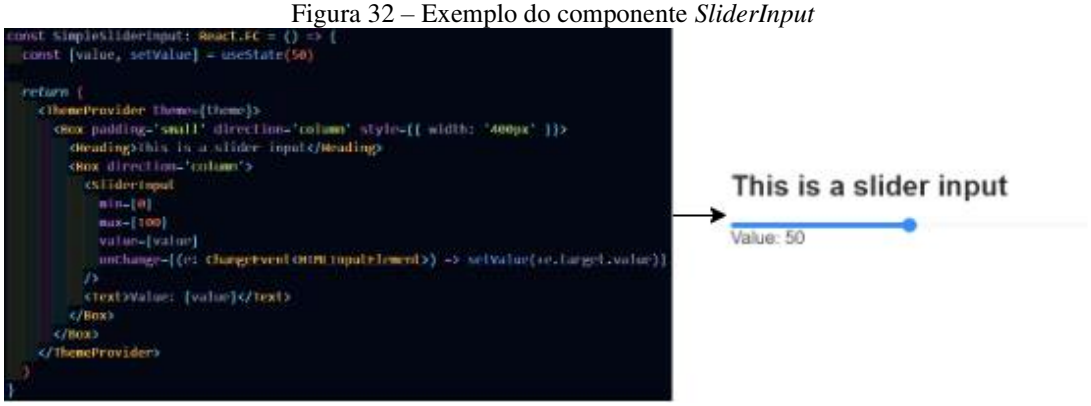
Figura 31 – Exemplo do componente *Select*

Fonte: Própria (2020)

Slider Input

É componente com a finalidade de receber um dado numérico. Porém, ao contrário do *Number Input*, esse input não recebe o seu valor a partir de digitação, mas sim por meio de uma barra de controle. Essa barra pode ser arrastada para direita e para esquerda, controlando o valor do *input*.

Figura 32 – Exemplo do componente *SliderInput*



Fonte: Própria (2020)

Table

Table é um componente responsável por realizar uma visualização de uma tabela. Geralmente é utilizado quando existe a necessidade de exibir uma grande quantidade de dados, mas sem consumir muito espaço.

Figura 33 – Exemplo do componente *Table*

Fonte: Própria (2020)

Table Body

Esse componente representa o conteúdo principal de uma tabela. É possível estilizar esse componente adicionando espaçamentos e trocando a cor de fundo.

Table Footer

Table Footer é o componente responsável por adicionar uma linha final na tabela. Essa linha final geralmente é usada para criar paginações ou até mesmo informações aglomeradas em uma tabela, como a soma de valores de uma coluna.

Table Head

É o componente responsável por representar a primeira linha da tabela que contém os títulos de cada coluna, de acordo com a estrutura de tabelas disponibilizado pelo HTML.

Table Heading

É o componente responsável por representar a uma célula dentro do *Table Head*. Esse componente tem a funcionalidade de estilizar o título das colunas, como adicionar tamanho e cores de texto diferente do conteúdo da tabela.

Table Item

TableItem representa uma célula de contida dentro do *TableRow*, de acordo com as especificações de tabela do HTML. Esse componente é o envoltório do dado a ser mostrado e pode ser customizado para alterar o visual, caso exista a necessidade.

Table Row

Esse componente representa uma linha tabela, sendo utilizado no *Table Body*. É possível configurar espaçamentos e cores, mudando o visual das linhas.

Text

Text é o componente base que cuida de visualizações de texto. Esse componente, por meio de propriedades, torna fácil a customização da tipografia da aplicação, como tamanho de texto, cores e alinhamento.

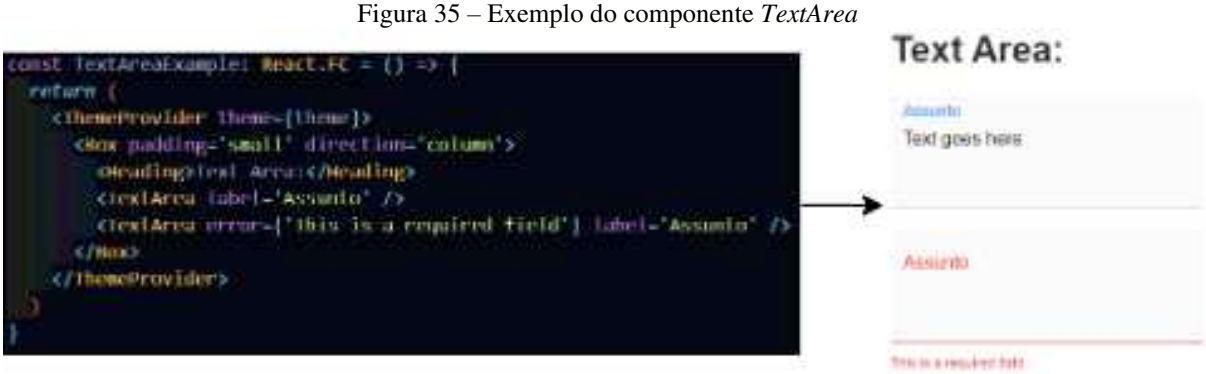
Figura 34 – Exemplo do componente *Text*

Fonte: Própria (2020)

TextArea

É um componente construído em cima do Input com funcionalidades e visuais parecidos com o *TextInput*, porém sua área de entrada de texto é bem maior, sendo mais bem utilizado quando existe a necessidade de receber textos longos. Como os demais *inputs*, todas as funcionalidades desse componente são customizáveis.

Figura 35 – Exemplo do componente *TextArea*

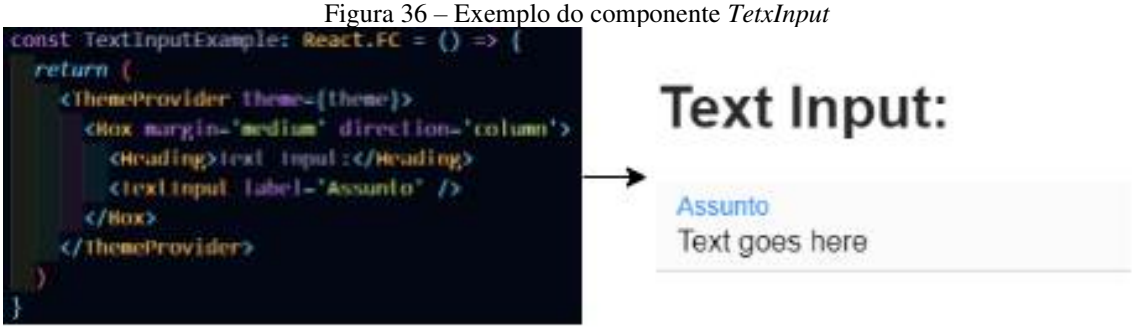


Fonte: Própria (2020)

TextInput

TextInput é um componente construído em cima do componente *Input*. Ele tem a funcionalidade de aceitar dados de texto. Ele possui diversas propriedades para customizar suas funcionalidades, facilitando a utilização.

Figura 36 – Exemplo do componente *TetxInput*

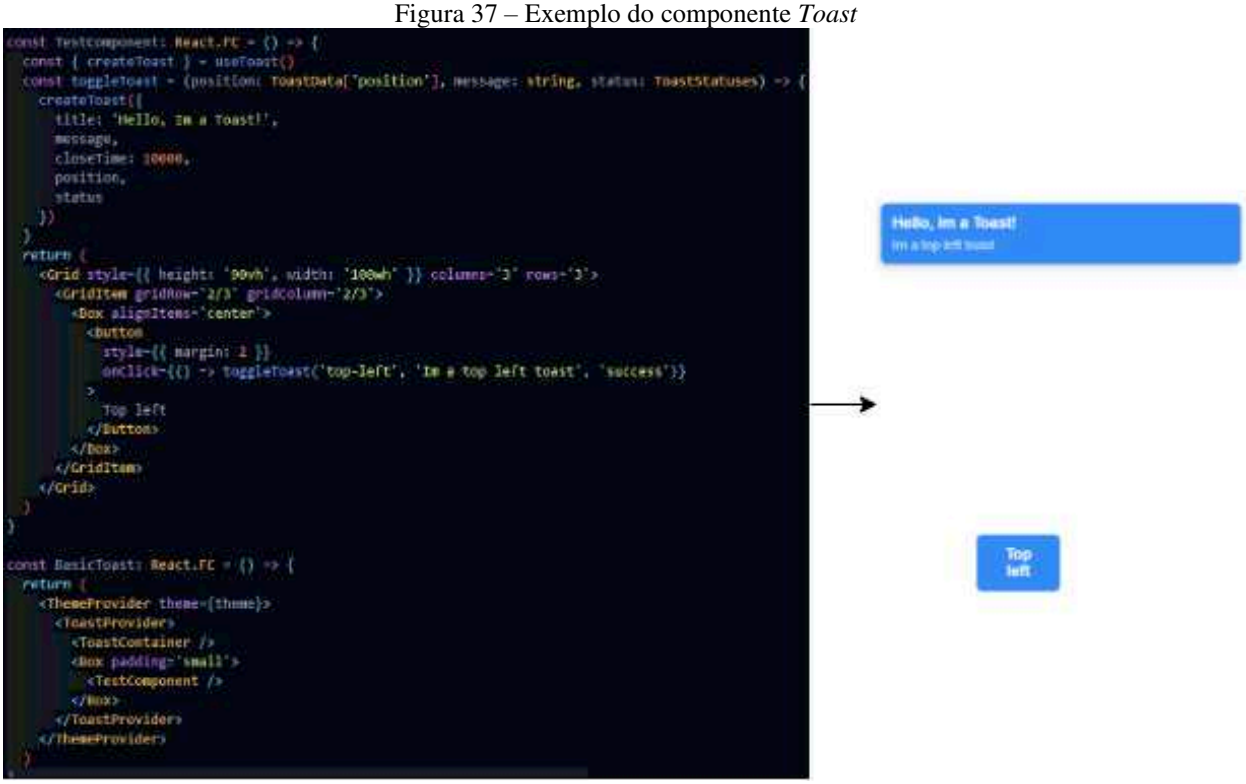


Fonte: Própria (2020)

Toast

Toast é um componente com a finalidade de gerar atualizações rápidas o usuário, sendo mostrado após um evento ou ação do usuário. *Toast* é uma pequena janela que pode aparecer em todas as extremidades da aplicação, suportando diferentes cores e conteúdo personalizável.

Figura 37 – Exemplo do componente *Toast*



Fonte: Própria (2020)

Helios

Helios é componente mais importante da biblioteca, sem ele, todo os componentes ficarão sem sua estilização, podendo gerar erros ao tentar utilizar a biblioteca, tornando sua utilização indispensável. Esse componente deve ser colocar na parte inicial da árvore de componentes da aplicação. Esse componente tem a responsabilidade de passar automaticamente, por meio de propriedades, o tema para todos os componentes da aplicação.

Tema é um objeto que contém definições de estilos para todos os componentes da biblioteca, se tornando o principal arquivo de configuração dos componentes. A biblioteca por padrão já possui um Tema criado, porém é possível criar um tema totalmente customizado para atender diferentes necessidades de estilo. Os estilos definidos pelo tema são espaçamentos, palheta de cores, sombreados, animações, tamanho e formato.