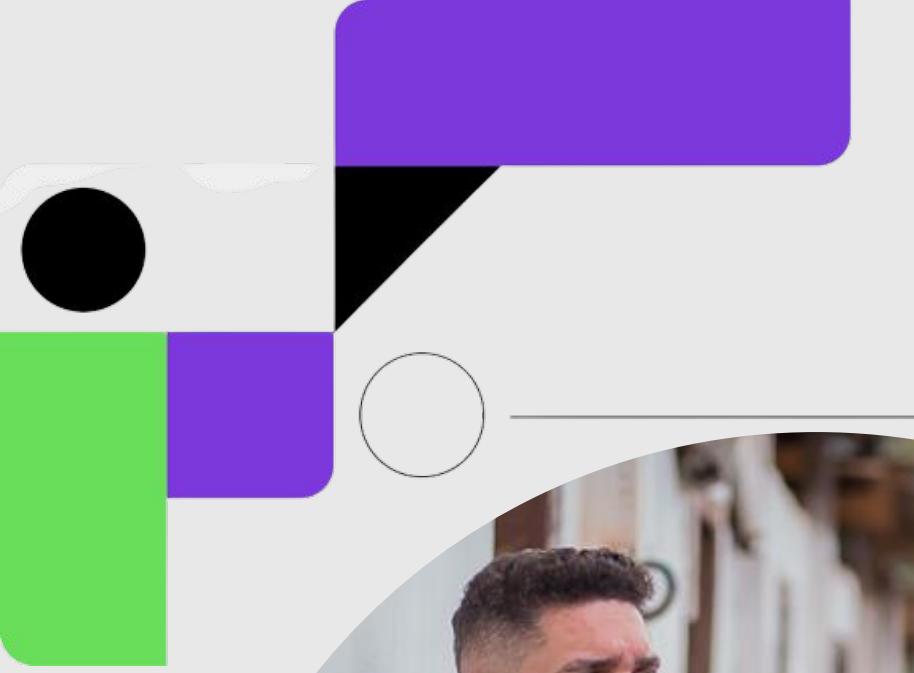


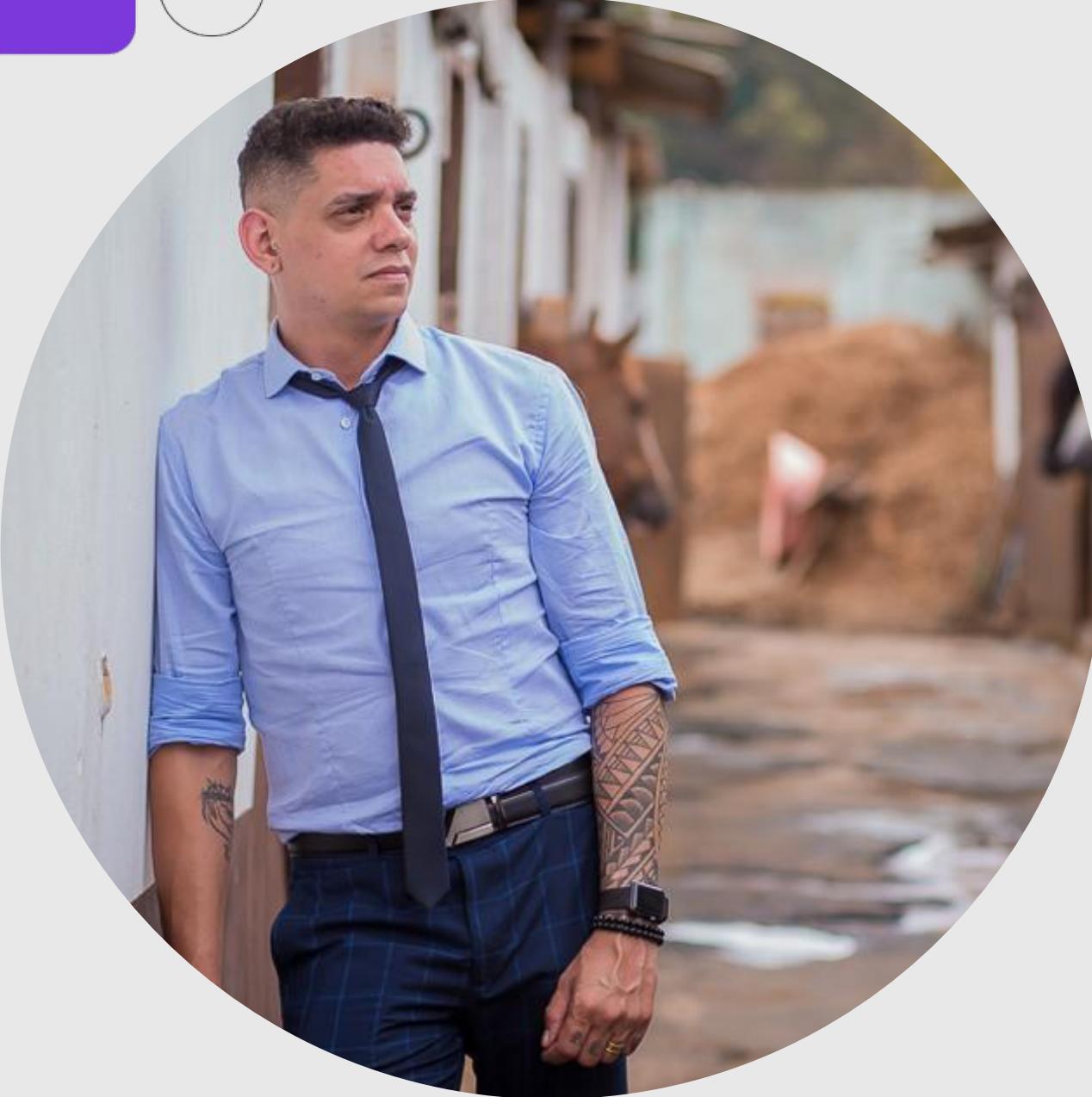


# React

*Material Complementar*



# Douglas Morais



34 anos, atua no mercado de desenvolvimento a cerca de 14 anos atualmente Techlead na Kenlo. Apaixonado por Tecnologia e todas as novidades do segmento.

Especialista em desenvolvimento frontend um eterno entusiasta do desenvolvimento mobile.



[linkedin.com/in/douglasmoraisdev/](https://linkedin.com/in/douglasmoraisdev/)



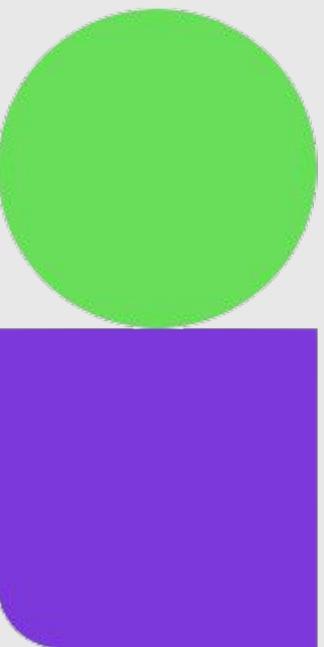
<https://github.com/mrdouglasmorais>

#PraCegoVer: Fotografia do autor Douglas  
Morais.

# Índice

- + [O que é o React?](#)
- + [Iniciando o projeto com TypeScript.](#)
- + [Sintaxe do TSX.](#)
- + [Class Components x Function Components.](#)
- + [Conhecendo os Hooks.](#)
- + [Trabalhando com Rotas.](#)
- + [Trabalhando com Props.](#)
- + [Conectando a API.](#)
- + [Publicando projeto na Vercel.](#)

# Introdução



Nesta apostila vamos aprender mais sobre o React, a proposta e o principal uso desta biblioteca que ganhou tanto espaço no mercado.

Para nossa jornada iniciaremos do "Zero" um projeto e vamos "dissecar" cada um funcionalidade e assuntos importantes em torno desta tecnologia. Trazendo conhecimentos importantes sobre as funcionalidades e práticas mais atuais, até a publicação do mesmo.



# O que é o React?

*Antes de iniciar nosso projeto vamos entender assuntos importantes em torno deste ecossistema.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# O que é o React?



**React** antes de qualquer coisa é uma biblioteca construtora de interfaces... Mas como assim uma biblioteca? Não seria um framework?

Não, o **React** é uma biblioteca e as maiores diferenças entre uma biblioteca e um framework é o fato de um framework te prender a uma gama de soluções fornecidas pelo mesmo. Enquanto uma biblioteca te permite ejetar o projeto e dar continuidades sem o uso da solução e este é o caso do React ele te permite ejetar o projeto caso você tenha a decisão de não utilizar mais a Lib permitindo dar continuidade usando **Javascript** ou **TypeScript**.

Seguindo nossa linha de raciocínio o **React** é uma biblioteca desenvolvida e mantida pelo time do Facebook, surgiu por volta de 2011 e hoje é uma das tecnologias mais utilizadas do mercado segundo o site *Stack Share* para visualizar clique [aqui](#).

Para saber mais, acesse a documentação oficial clique [aqui](#).

# Onde usar o React?



Antes de tudo, no mundo da tecnologia não existe uma solução bala de prata. Levando isto em consideração vamos entender onde e quando utilizar o **React** em seus projetos.

O **React** surgiu com o objetivo de otimizar a atualização e a sincronização de atividades simultâneas como: feed de notícias, interações com chat e afins e tudo isso sem a necessidade de recarregar a aplicação, tudo isso é feito através do gerenciamento de estado, onde qualquer alteração realizada atualiza a interface.

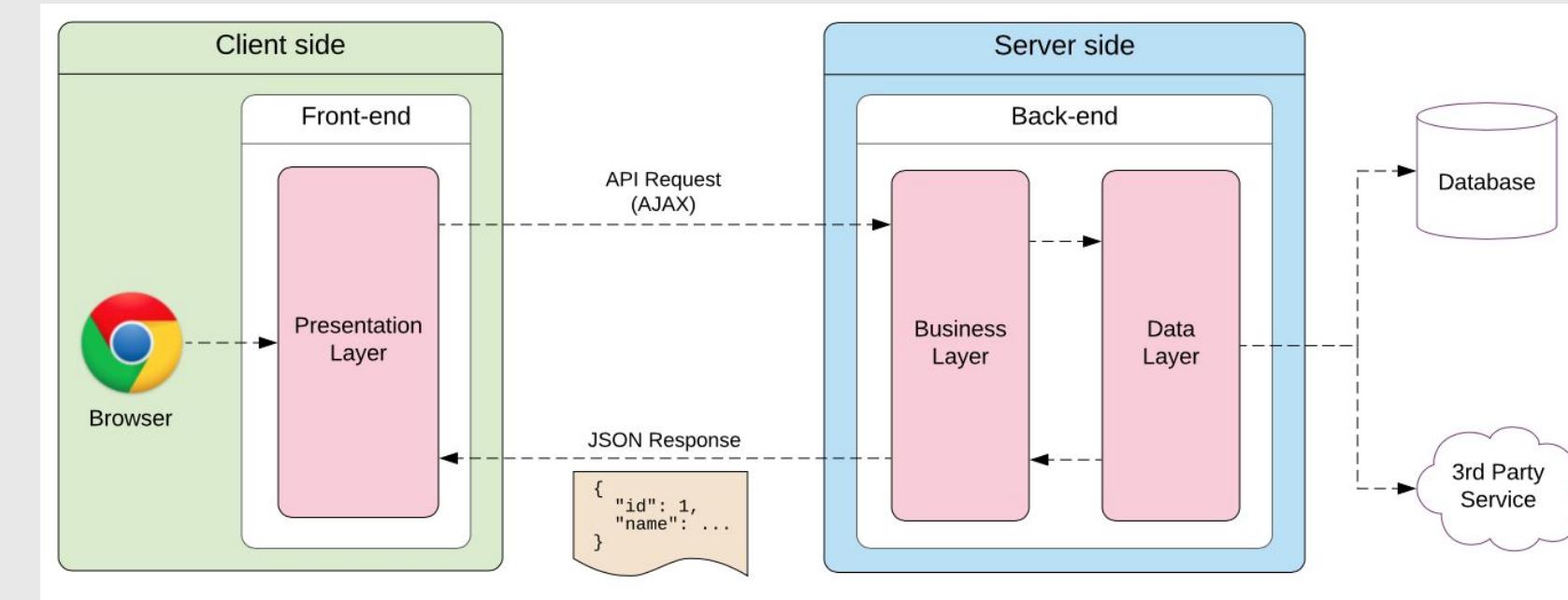
Muito bem! mas eu posso utilizar o **React** para construção de websites? Até pode ser utilizado em soluções como e-commerce que dependem de otimização de **SEO** e afins, porém torna-se muito oneroso, atualmente o mercado tem soluções mais adequadas para foco em performance como é o caso do **NextJS** um framework do **React**.

# O que é um SPA?

**SPA** (Single Page Application) é uma aplicação **WEB** onde todos os elementos e rotas são tratadas em uma página única.

Um exemplo muito prático é o **Gmail** do **Google** ele é um **SPA** no modelo **SaaS** (Software as a Service ou Software como Serviço). Ao acessar seus emails tudo que for necessário de interações como: Ler, enviar emails ou até mesmo responder são realizados na mesma página, sem a necessidade de abrir novas abas no browser para cada ação.

Ao lado temos um diagrama ilustrando o funcionamento de uma **SPA**.



#PraCegoVer: Diagrama de solução SPA formato Client Side Rendering



# Iniciando projeto com React

#PraCegoVer: Dois notebooks  
o primeiro apresentado o teclado  
o segundo apresentando o  
monitor.

*Agora é hora de colocar a mão na massa  
e iniciar o nosso projeto.*

# Introdução

Agora é a hora de iniciar o nosso projeto e antes de "colocar a mão na massa" nós vamos precisar ter algumas dependências e softwares instalados em nosso equipamento.

O **React** depende da runtime do **NODEJS** e caso você não tenha instalado clique aqui para instalar em seu equipamento.

Para trabalhar em nosso código é importante utilizar um editor de sua preferência, no meu caso eu utilizo o **VSCODE** (Visual Studio Code), para instalar em seu equipamento clique aqui.

Para acompanhar o fluxo e evolução do projeto nós vamos precisar versionar o nosso código e para isso é necessário ter o Git devidamente instalado no equipamento com GitHub configurado (estou disponibilizando o projeto em meu GitHub para que você possa acompanhar passo a passo para visualizar clique [aqui](#)).

\***Atenção** a versão mais atualizada do **NODEJS** é a versão 16.14.2 (LTS) aconselho utilizar a versão mencionada ou acima da 14.x.x

# Iniciando um projeto



Para iniciar o nosso projeto nós faremos uso de um boiler plate ou seja de um "start engine" chamado *Create React App* com **TypeScript** (Super set do javascript), lembrando que existem várias maneiras de iniciar o seu projeto eu particularmente utilizo o método mais completo onde as configurações de dependências e **WEBPACK** são feitas manualmente.

Para mais detalhes do *Create React App* clique [aqui](#) para ver a documentação oficial.

Vamos utilizar o gerenciador **NPX** desta maneira podemos ter em nosso projeto a versão mais atual do **React** sem a necessidade de instalar globalmente em nosso equipamento.

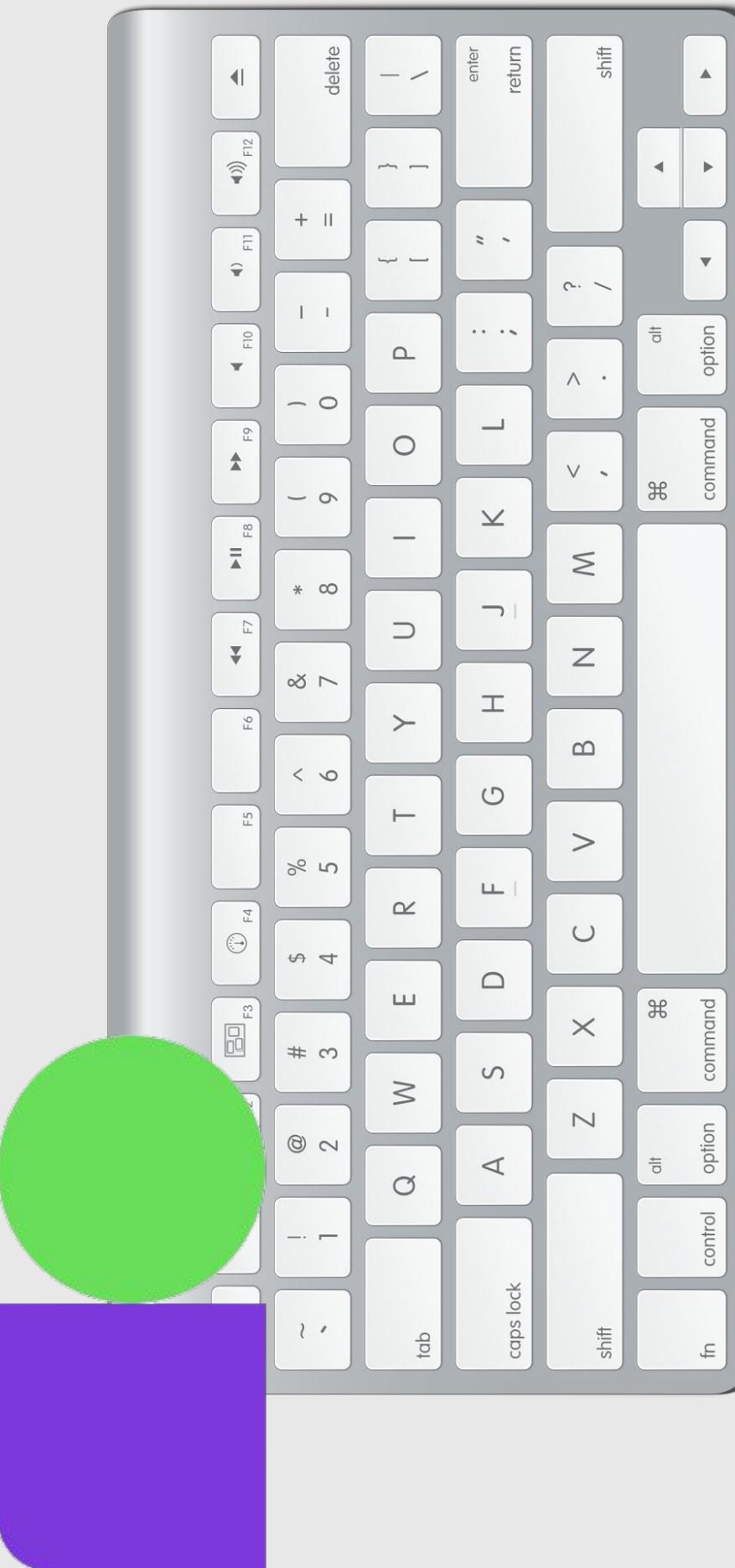
Então vamos adiante.

\***Atenção** para cada trecho do projeto, vou disponibilizar commits no repositório com as devidas atualizações.

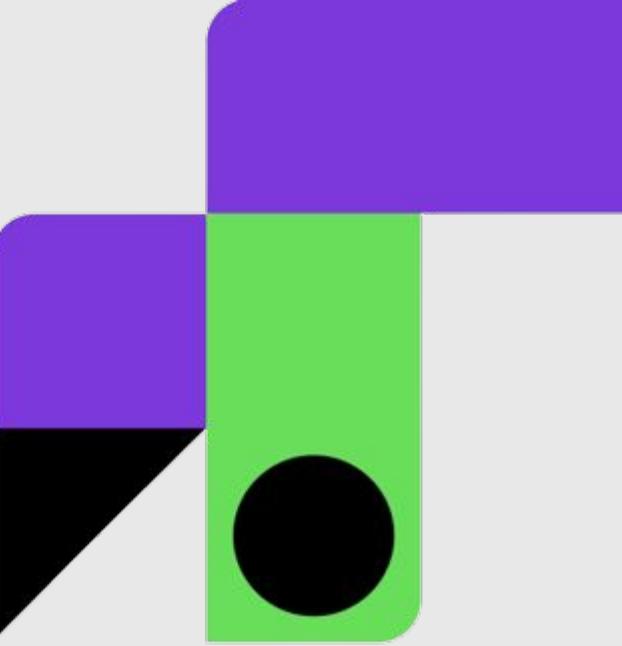
# Sobre o React

Neste módulo aprendemos conteúdos conceituais em torno deste ecossistema de soluções frontend altamente poderoso e versátil.

Conceitos de como onde e como aplicar determinadas soluções são importantíssimos principalmente em momentos de concepção de novos produtos digitais.



# Iniciando um projeto



Abra o terminal de sua máquina e se localize no diretório onde quer armazenar o seu projeto.

Estou usando o sistema operacional **MacOS** e os comandos são muito similares aos do **Linux**.

Devemos instruir o seguinte comando no terminal.

```
→ npx create-react-app gama-react-ts --template typescript
```

## O que estamos instruindo:

**npx**: Gerenciador de dependências.

**create-react-app**: O pacote que estamos invocando.

**gama-react-ts**: O nome do nosso projeto.

**--template**: Flag de template adicional.

**typescript**: O template utilizado.

A screenshot of a macOS terminal window titled "Desktop — douglasmoraes@Douglas-MacBook-Pro — ~/desktop — zsh — 80x24". The window shows the command "npx create-react-app gama-react-ts --template typescript" being typed. The terminal has a dark theme with purple decorative lines at the top.

#PraCegoVer: Terminal com a instrução de comando para início do projeto: npx create-react-app gama-react-ts –template typescript.

# Iniciando um projeto



Após o processo de instalação o terminal informará se instalou ou não a aplicação, conforme a imagem ao lado.

Em seguida instrua os seguintes comandos:

```
douglasmorais in ~/desktop took 36s
→ cd gama-react-ts
douglasmorais in gama-react-ts on master
→ code .
```

## O que estamos instruindo:

**cd gama-react-ts**: Acessar a pasta do projeto.

**code .**: Abrir projeto com VSCode.

```
yarn start
  Starts the development server.

yarn build
  Bundles the app into static files for production.

yarn test
  Starts the test runner.

yarn eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd gama-react-ts
  yarn start

Happy hacking!
★ Done in 35.80s.
douglasmorais in ~/desktop took 36s
→ cd gama-react-ts
douglasmorais in gama-react-ts on [master]
→ via ● v16.13.0 code .
```

#PraCegoVer: Terminal com a instrução de projeto instalado com sucesso, além dos comandos de acessar pasta e abrir projeto com VSCode.

# Estrutura de arquivos

Agora vamos entender a nossa estrutura de arquivos.

*Diretórios:*

**node\_modules**: Diretório onde armazena todas as dependências de nosso projeto.

**public**: Centralizar arquivos responsáveis pela compilação do nosso projeto.

**src**: Diretório onde fica o código fonte do projeto.

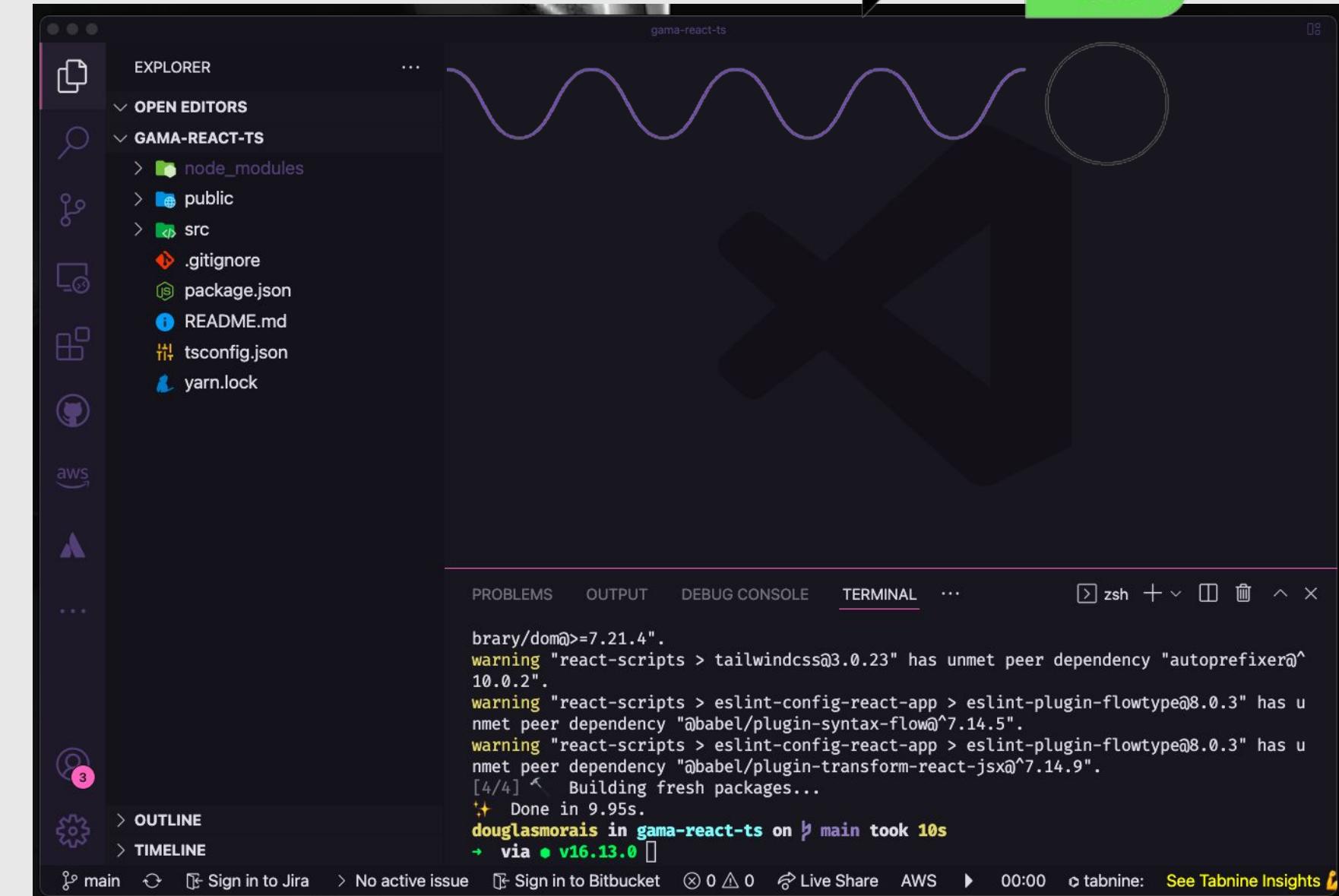
*Arquivos:*

**.gitignore**: Arquivos e diretórios a serem ignorados pelo github.

**package.json**: Configurações do projeto como nome e dependências de desenvolvimento e produção.

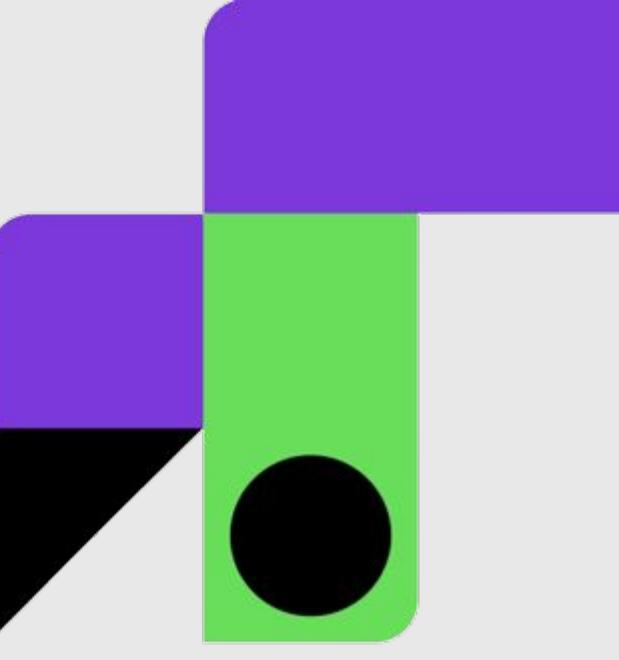
**README.md**: o Leia-me de nosso projeto no github

**tsconfig.json**: Configurações de sintaxe e ajustes do Typescript.



#PraCegoVer: VSCode com projeto aberto e estrutura de pastas amostra no box a esquerda.

# Iniciando em desenvolvimento



Para validar o funcionamento do nosso projeto, vamos iniciar em ambiente de desenvolvimento e para isto adicione o seguinte comando.

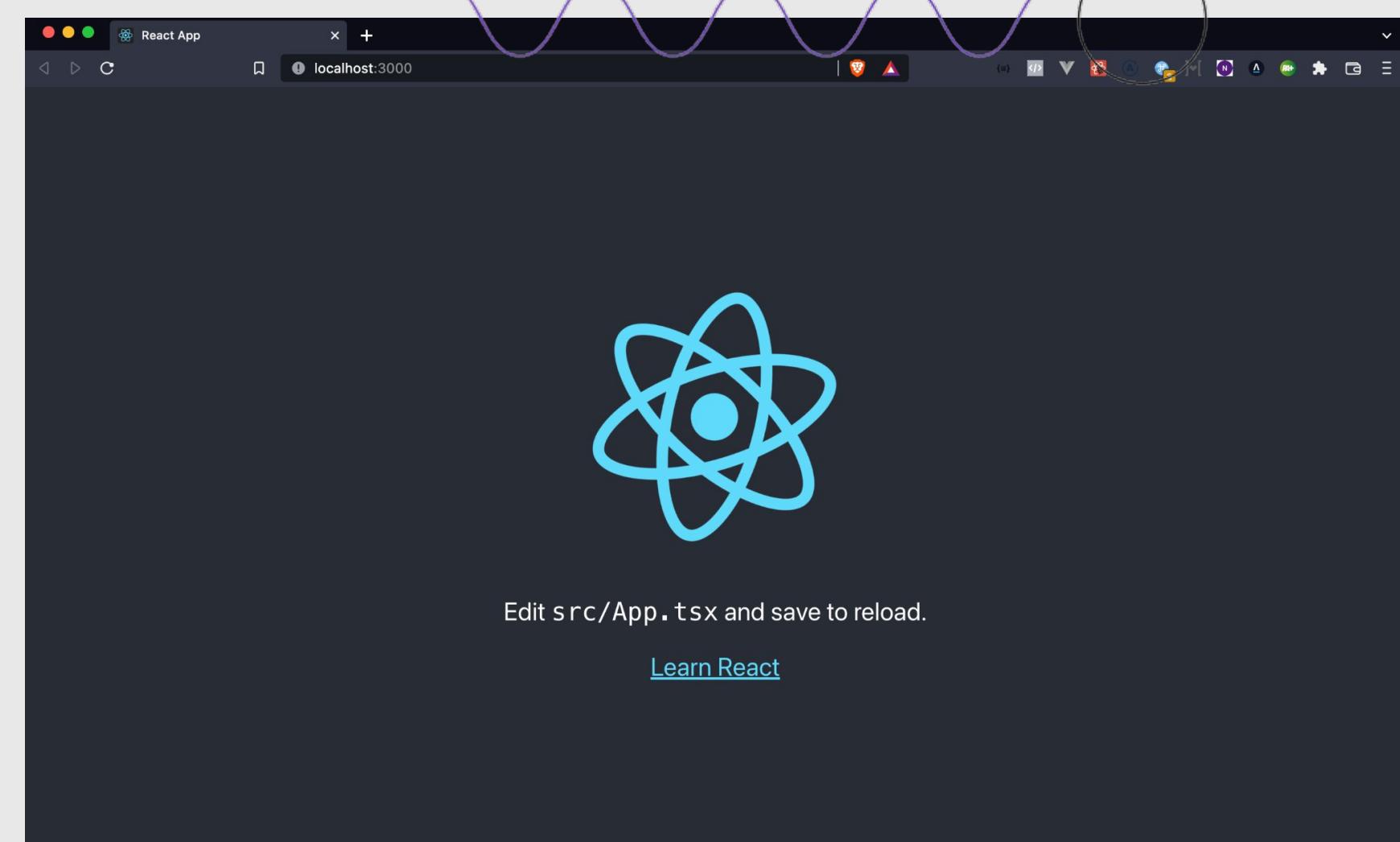
```
→ yarn start
```

Em seguida uma janela do seu navegador vai abrir no seguinte endereço:

<http://localhost:3000/>

O projeto deve aparecer em seu navegador conforme a imagem ilustrada ao lado.

\*Caso não abra automaticamente você pode inserir direto no navegador manualmente.



#PraCegoVer: Navegador com nosso projeto iniciado.

# Iniciando com React

Neste módulo tivemos uma breve introdução ao **React** fazendo uso do create-react-app e iniciando o projeto em nosso ambiente local.

Lembrando que existem inúmeras maneiras de "iniciar um projeto do zero" a maneira que iniciamos o nosso é a mais rápida a mérito de aprendizado.

Não se esqueçam existem várias e várias maneiras de implementar **React** em projeto neste módulo aprendemos a criar um projeto dedicado.





# Sintaxe do TSX

*Componentes React com TypeScript*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Introdução



Agora é hora de aprender sobre a sintaxe do *TSX*, mas no final das contas o que é?

Basicamente a forma com que o **React** interpreta os templates implementados no projeto porém no formato do **TypeScript** sendo assim todos os arquivos que carregam elementos a serem renderizados devem ter a extensão *.tsx* .

O que de fato compõem esta sintaxe?

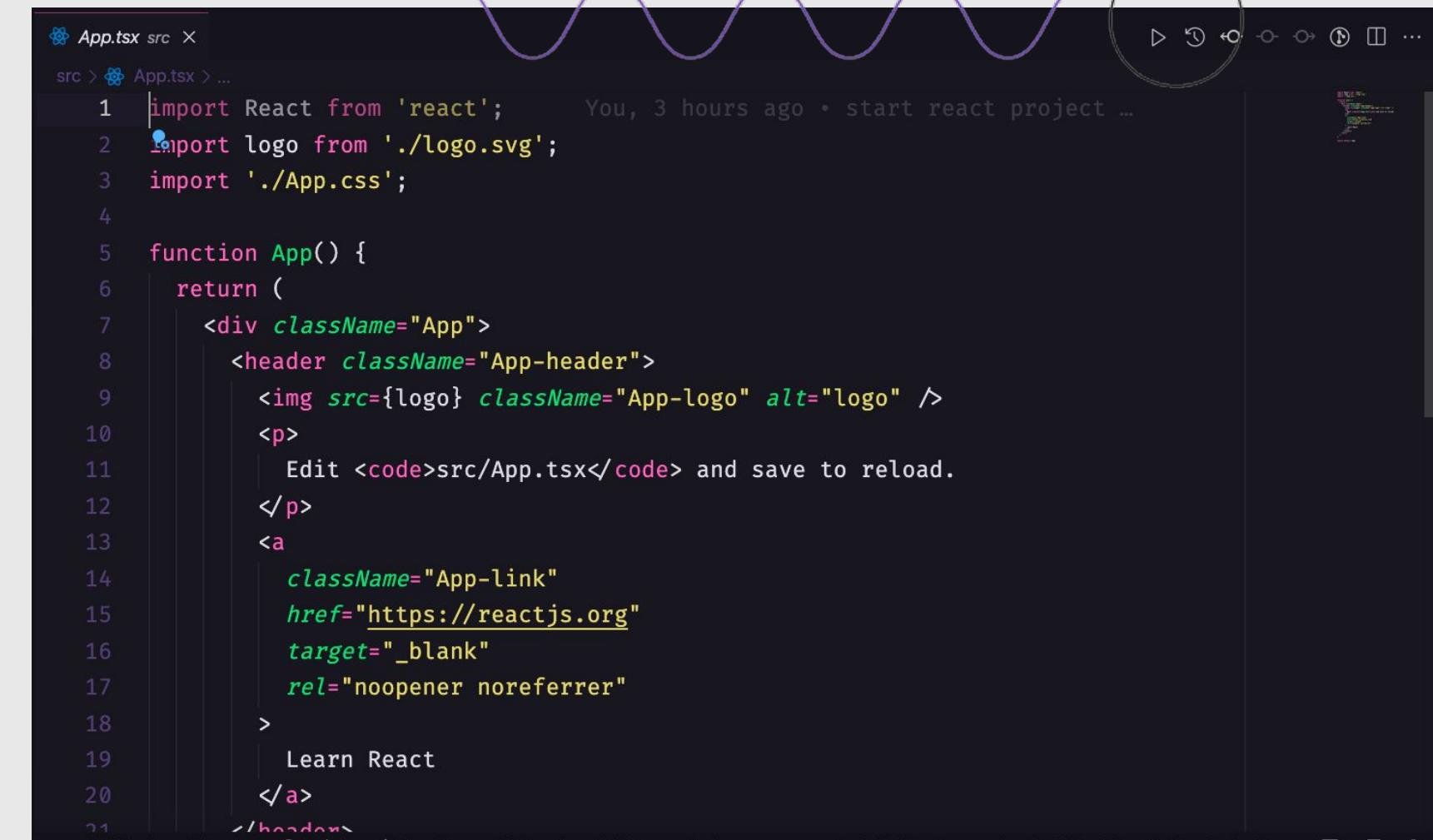
No caso da função, ela retorna um elemento muito similar ao HTML porém com a permissividade de criar novas TAGs de acordo a organização e componentização estrutural de seu projeto, a seguir vamos ter exemplos mais prático em torno desta sintaxe.

# Sintaxe TSX

Para este momento vamos precisar acessar através do VSCode ou editor de sua preferência o seguinte caminho `src/App.tsx` após abrir o arquivo você vai se deparar com a sintaxe igual a imagem ao lado.

A seguir temos um exemplo de componente:

```
import React from 'react';
// importa do React dependências
const App: React.FC = () => {
  // O React.FC functional component e a tipagem do TS
  return (
    <div>
      <p>Olá</p>
    </div>
  );
}
// Uma arrow function retornando "Olá"
export default App;
// Exportando o componente para todo projeto
```

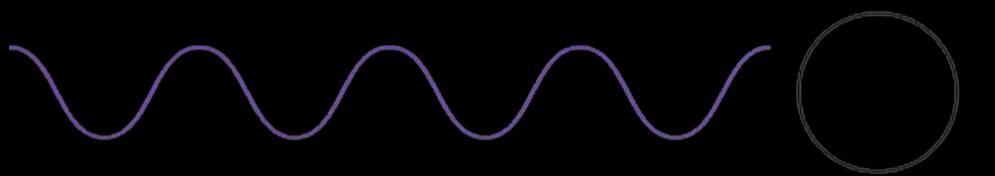


```
App.tsx src ×
src > App.tsx > ...
1 import React from 'react'; You, 3 hours ago • start react project ...
2 import logo from './logo.svg';
3 import './App.css';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="logo" />
10        <p>
11          Edit <code>src/App.tsx</code> and save to reload.
12        </p>
13        <a
14          className="App-link"
15          href="https://reactjs.org"
16          target="_blank"
17          rel="noopener noreferrer"
18        >
19          Learn React
20        </a>
21      </header>
```

#PraCegoVer: VSCode com arquivo App.tsx aberto.

# Anatomia do TSX

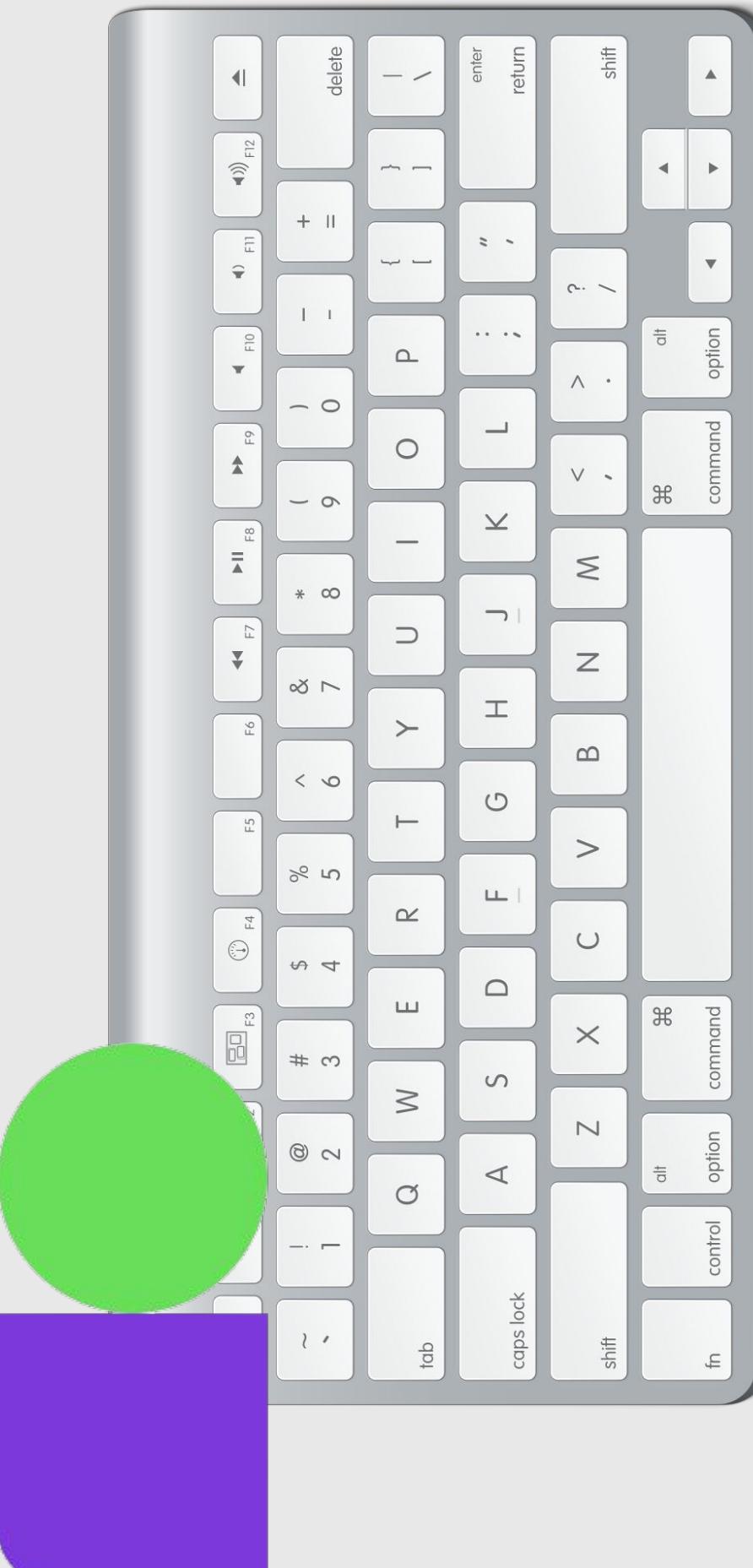
```
import React from 'react';
// importa do React dependências
// Todas as importações de componentes ou até mesmo arquivos devem ser feita neste trecho
const App: React.FC = () => {
  // O React.FC functional component e a tipagem do TS
  function hello(){
    alert('Olá!!!')
  }
  // funções adicionais ou até mesmo os Hooks devem estar neste trecho antes do retorno de nosso componente
  return (
    <div>
      <button
        onClick={hello}
        // Evento de clique recebendo uma função
        >Clique aqui</button>
    </div>
    // Os retornos contendo HTML devem estar neste trecho
  );
}
// Uma arrow function retornando um botão escrito clique aqui disparando um evento de clique com um alert no browser escrito "Olá!!!"
export default App;
// Exportando o componente para todo projeto
```



# Sintaxe do TSX

Neste módulo entendemos um pouco mais sobre a sintaxe do TSX, e o principal como podemos criar "componentes do zero" iniciando manualmente uma estrutura.

Nos módulos a seguir teremos a compreensão formatos e métodos de trabalhar com esta sintaxe e como podemos importar funcionalidades desenvolvidas por nós mesmos ou qualquer outra como arquivos de CSS e afins.



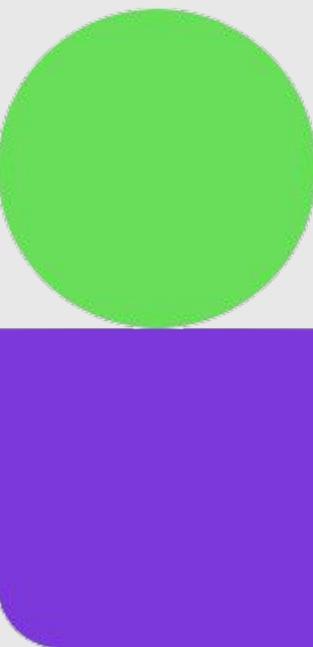


# CC x FC

*Class Components x Functional Components*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Introdução



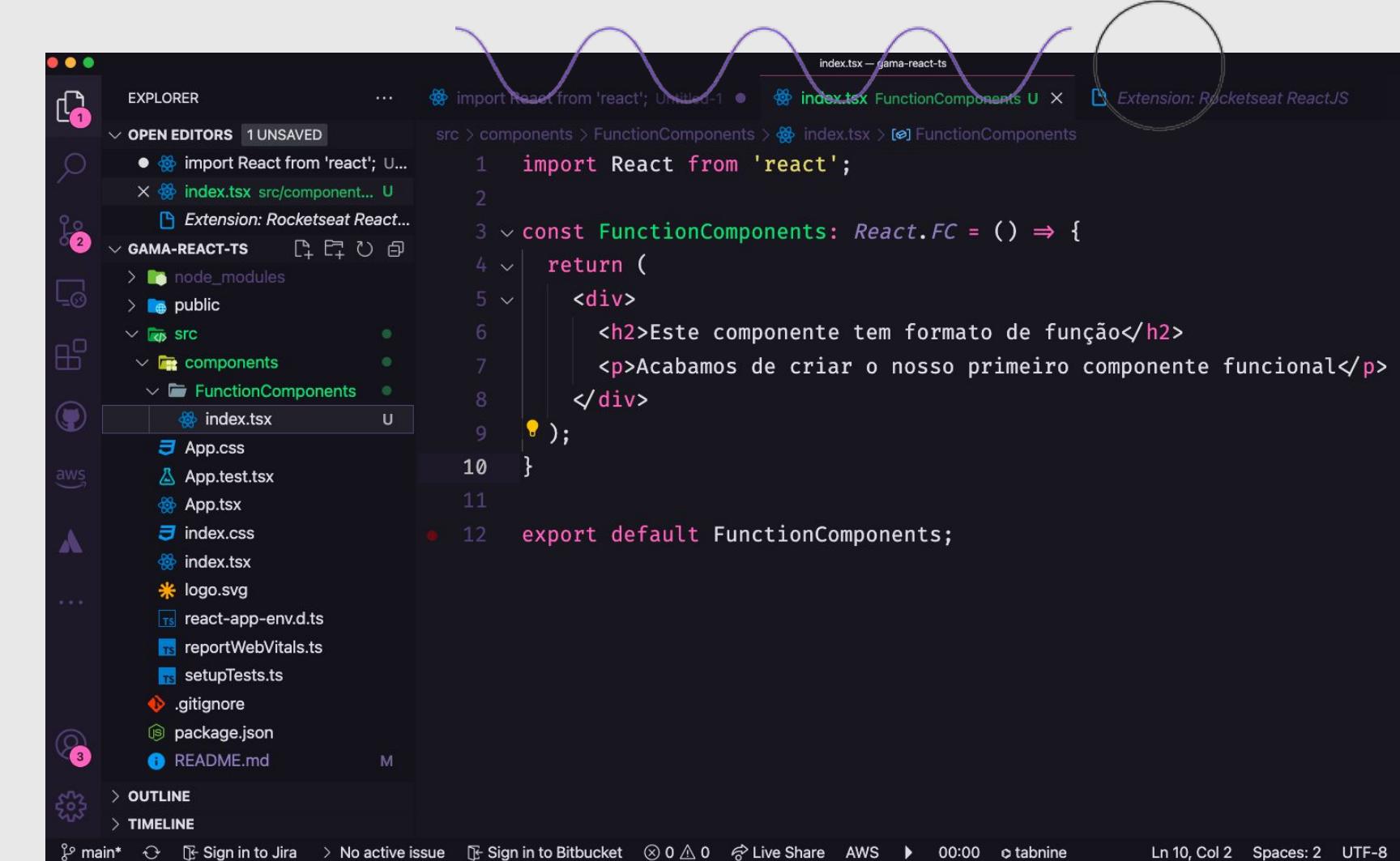
Neste módulo vamos aprender as principais diferenças entre componentes no formato de Função e no formato de classes, vale a pena enfatizar que o uso de classes está em desuso ou seja não é possível adicionar hooks a uma classe.

Para este projeto estamos trabalhando com o React na versão 17.2.1 e com esta atualização temos algumas peculiaridades que mencionaremos no decorrer de cada implementação.

# Function component

Para iniciar o nosso componente, vamos criar uma nova pasta dentro de `src` e vamos chamar de `components` ou seja nosso caminho relativo vai ser `src/components` a seguir vamos criar mais um diretório chamado de `FunctionComponents` com arquivo `index.tsx` e o caminho final será o seguinte: `src/components/FunctionComponents/index.tsx`

Nos passos a seguir vamos implementar a estrutura de código e a descrição de trechos estarão nos comentários.



The image shows a screenshot of the Visual Studio Code (VSCode) interface. On the left, the Explorer sidebar displays the project structure:

- `OPEN EDITORS`: `import React from 'react'; U...`, `index.tsx src/component...` (marked with a red circle), `Extension: Rocketseat React...`
- `GAMA-REACT-TS`: `node_modules`, `public`, `src` (marked with a green circle), `components` (marked with a blue circle), `FunctionComponents` (marked with a yellow circle), `index.tsx`
- `aws`, `App.css`, `App.test.tsx`, `App.tsx`, `index.css`, `index.tsx`, `logo.svg`, `react-app-env.d.ts`, `reportWebVitals.ts`, `setupTests.ts`, `.gitignore`, `package.json`, `README.md`
- `OUTLINE`, `TIMELINE`

In the center, the code editor shows the `index.tsx` file content:

```
1 import React from 'react';
2
3 const FunctionComponents: React.FC = () => {
4   return (
5     <div>
6       <h2>Este componente tem formato de função</h2>
7       <p>Acabamos de criar o nosso primeiro componente funcional</p>
8     </div>
9   );
10 }
11
12 export default FunctionComponents;
```

At the bottom of the interface, there are several status icons and text: "main\*", "Sign in to Jira", "No active issue", "Sign in to Bitbucket", "0 ▲ 0", "Live Share", "AWS", "00:00", "tabnine", "Ln 10, Col 2", "Spaces: 2", "UTF-8".

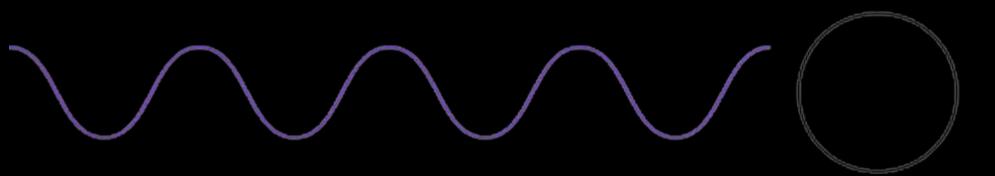
#PraCegoVer: VSCode com arquivo `index.tsx` aberto.

# Function component

```
// Importação do React
import React from 'react';

// Declarando a nossa função, pode ser formato arrow ou o convencional exemplo abaixo
// Function FunctionComponents(){ ... }
const FunctionComponents: React.FC = () => {
  const name = 'função'
  // armazenando um valor em uma string para consumir no nosso código
  return (
    <div>
      <h2>Este componente tem formato de {name}</h2>
      {/* para utilizar valores dinâmicos informe dentro de {} */}
      <p>Acabamos de criar o nosso primeiro componente funcional</p>
    </div>
  );
}

export default FunctionComponents;
```

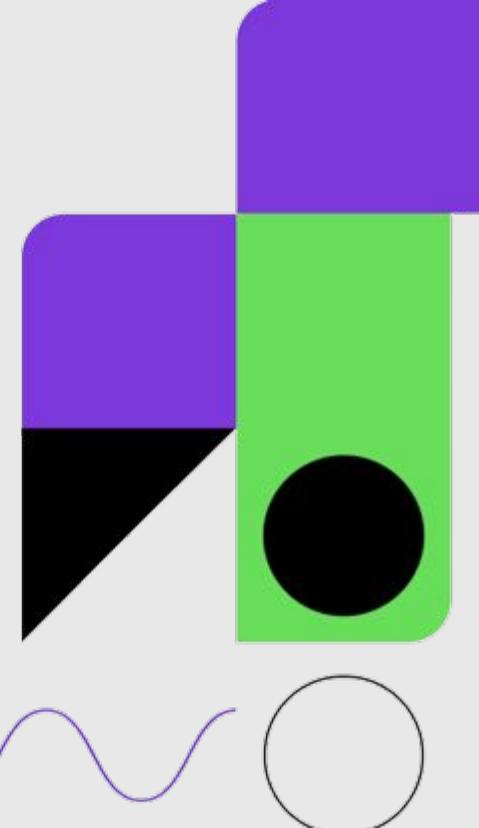


# Class component

Para iniciar o nosso componente, vamos criar uma nova pasta dentro de `src` e vamos chamar de `components` ou seja nosso caminho relativo vai ser `src/components` a seguir vamos criar mais um diretório chamado de `ClassComponents` com arquivo `index.tsx` e o caminho final será o seguinte:  
`src/components/ClassComponents/index.tsx`

Nos passos a seguir vamos implementar a estrutura de código e a descrição de trechos estarão nos comentários.

\*Importante enfatizar que este método de declarar componentes está em desuso.



```
index.tsx -- gama-react-ts
import React from 'react'; Untitled-1 ● index.tsx FunctionComponents U ● index.tsx ClassComponents U X App.tsx src M
src > components > ClassComponents > index.tsx > ClassComponents > render
1 // Importação do React
2 import React from 'react';
3
4 class ClassComponents extends React.Component{
5
6 // método de renderização da classe de
7 render() {
8 const name = 'classe'
9 // armazenando um valor em uma string para consumir no nosso código
10 return(
11   <div>
12     <h2>Este componente tem formato de {name}</h2>
13     /* para utilizar valores dinâmicos informe dentro de {} */
14     <p>Acabamos de criar o nosso primeiro componente formato de classe</p>
15   </div>
16 )
17 }
18 }
19
20 export default ClassComponents;
```

The screenshot shows a Visual Studio Code interface. The Explorer sidebar on the left displays a project structure with folders like 'GAMA-REACT-TS', 'node\_modules', 'public', and 'src'. Inside 'src', there are 'components' and 'ClassComponents' folders, each containing an 'index.tsx' file. The 'FunctionComponents' folder is also visible. The 'index.tsx' file in the 'ClassComponents' folder is open in the editor, showing code for a class-based React component named 'ClassComponents'. The code includes imports for React, a class definition extending 'React.Component', a render method that creates an H2 element with a dynamic value, and a final p element. The code is annotated with comments explaining the purpose of each part. The status bar at the bottom shows various VSCode settings and file information.

#PraCegoVer: VSCode com arquivo  
`src/components/ClassComponents/index.tsx` aberto.

# Class component

```
// Importação do React
import React from 'react';

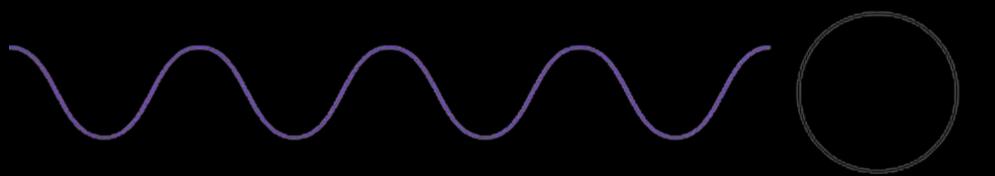
class ClassComponents extends React.Component{

    // método de renderização da classe
    render() {
        const name = 'classe'
        // armazenando um valor em uma string para consumir no nosso código

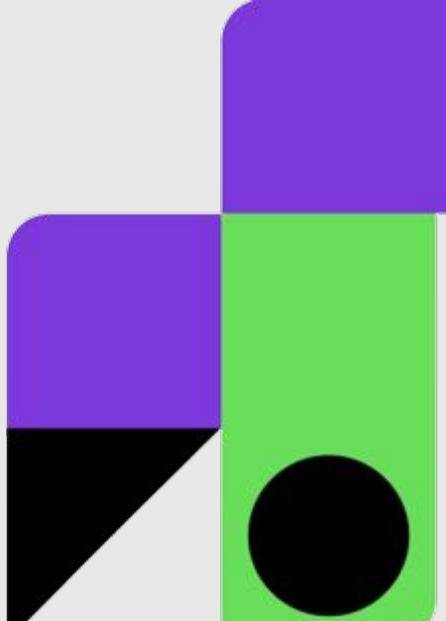
        // retorno de nosso método
        return(
            <div>
                <h2>Este componente tem formato de {name}</h2>
                {/* para utilizar valores dinâmicos informe dentro de {} */}
                <p>Acabamos de criar o nosso primeiro componente formato de classe</p>
            </div>
        )
    }

}

export default ClassComponents;
```



# Class component



Agora vamos importar os dois componentes que criamos em nosso arquivo App.tsx dentro de src.

E para validar o funcionamento vamos reescrever o componente e importando no formato correto de TAG importante ter claro que todo component **React** sem exceção deve iniciar com letra maiúscula, assim a estrutura consegue diferenciar elementos HTML de componentes.

Na próxima página teremos o arquivo estruturado e a descrição nos comentários.

The screenshot shows the VSCode interface with the following details:

- Explorer View:** Shows the project structure with files like index.tsx, App.tsx, and various test and configuration files.
- Open Editors:** Shows two unsaved files: index.tsx (FunctionComponents) and index.tsx (ClassComponents).
- Code Editor:** Displays the code for the ClassComponents/index.tsx file:

```
// Importação do React
import React from 'react';

class ClassComponents extends React.Component{
    // método de renderização da classe de
    render() {
        const name = 'classe'
        // armazenando um valor em uma string para consumir no nosso código
        return(
            <div>
                <h2>Este componente tem formato de {name}</h2>
                /* para utilizar valores dinâmicos informe dentro de {} */
                <p>Acabamos de criar o nosso primeiro componente formato de classe</p>
            </div>
        )
    }
}

export default ClassComponents;
```
- Bottom Status Bar:** Shows information like "Ln 17, Col 4", "Spaces: 2", "UTF-8", "LF", "TypeScript React", and other system status.

#PraCegoVer: VSCode com arquivo  
src/components/ClassComponents/index.tsx aberto.

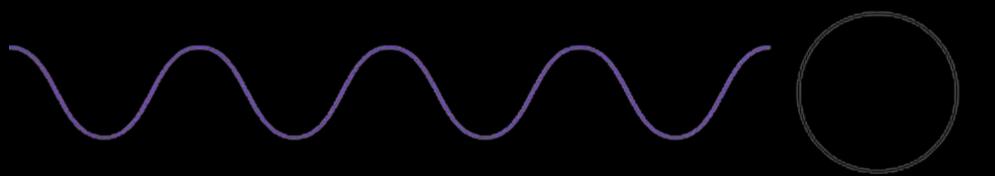
# Importando

```
// importação do React
import React from 'react';

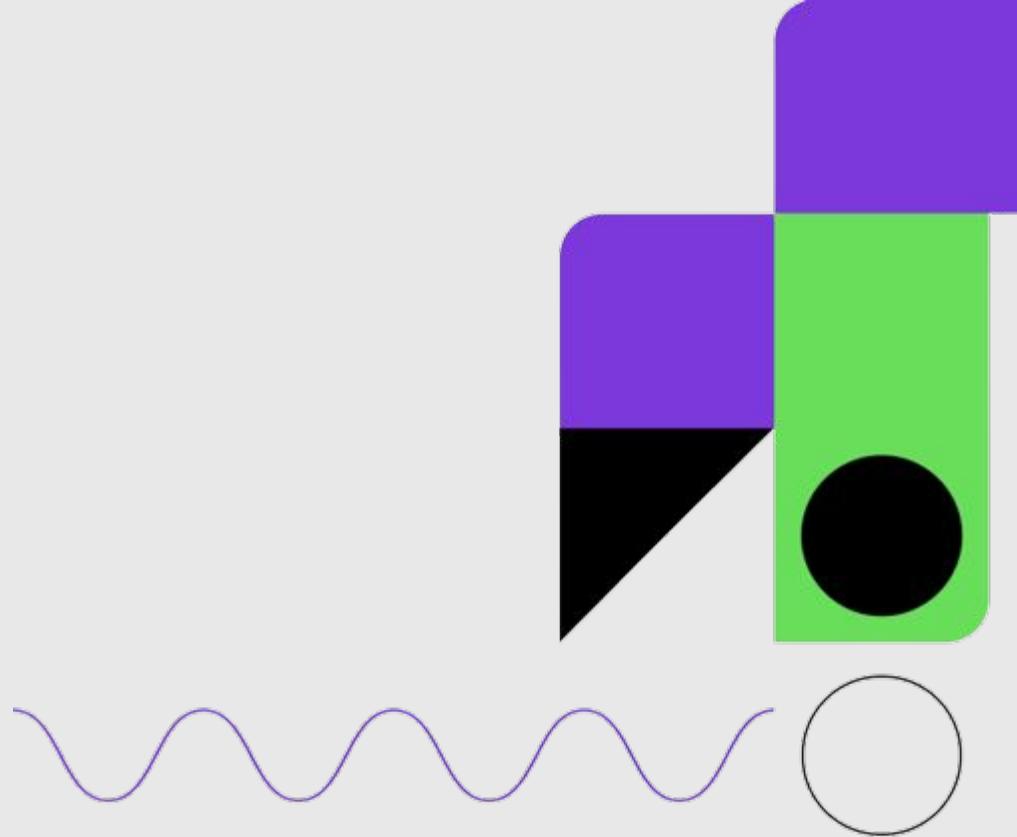
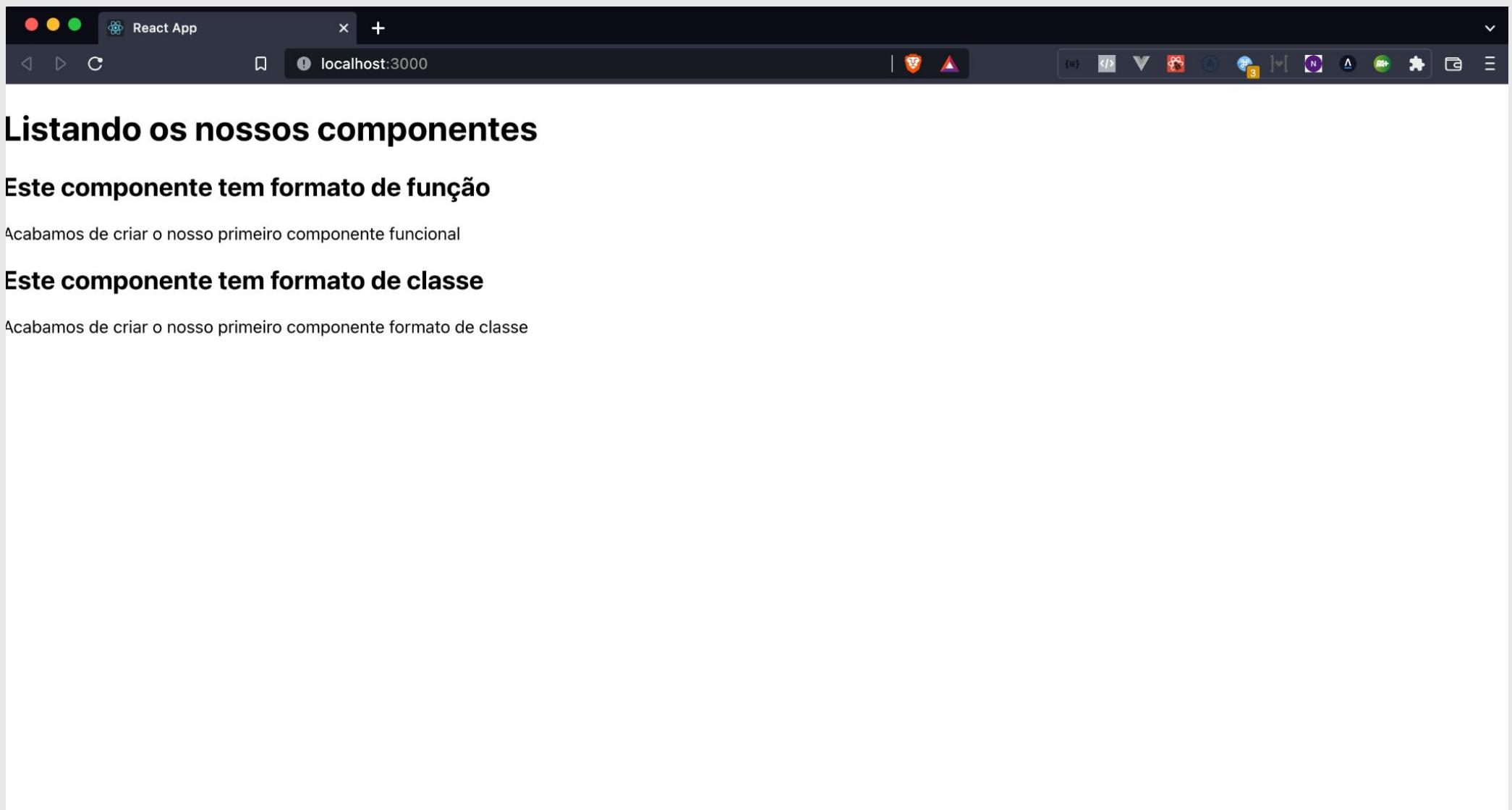
// importação dos componentes
import FunctionalComponent from './components/FunctionComponents';
import ClassComponents from './components/ClassComponents';

// componente no formato de função
const App: React.FC = () => {
  return(
    <div>
      <h1>Listando os nossos componentes</h1>
      <FunctionalComponent/>
      <ClassComponents/>
      {/* importando adicionando ao código os componentes */}
    </div>
  );
}

export default App;
// exportação
```



# Resultado final



Este é o resultado final do nosso aprendizado, clique [aqui](#) para acompanhar o commit no GitHub.

# CC x FC

Neste módulo aprendemos como elaborar componentes no formato de função e no formato de classe além de importar nossos componentes e implementar em nosso projeto.



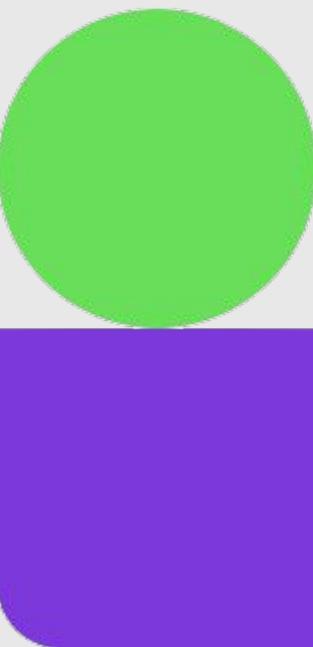


# Conhecendo os Hooks

*O que são os React Hooks e para que servem.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Introdução



Para este módulo, nós vamos entender e fazer uso de uma das funcionalidades mais poderosas do **React** os *Hooks*, mas no final das contas o que são os *Hooks*?

Os *Hooks* são funções que permite controlar e gerenciar os estados de nossa aplicação além de fornecer controles para controle de ciclo de vida de componentes, saiba mais sobre estados e ciclo de vida clicando [aqui](#).

Sendo assim, são essenciais para o desenvolvimento levando em consideração as práticas mais atuais.

Agora vamos à prática.

# Hooks

Entre os mais utilizados temos os seguintes Hooks:

**useState**: Responsável por setar e servir valores para nossa aplicação.

**useEffect**: Responsável por iniciar, observar ou destruir os estados de nosso componentes.

**useCallback**: Responsável por memorizar em nossa aplicação uma função, garantindo reutilizar e reduzir a processamento de nosso projeto.

**useRef**: Uma função que retorna um objeto mutável ou seja assim que uma alteração é passada ele armazena a referência como initial value persistindo em todo ciclo de vida da aplicação.



**useMemo**: Principal solução....

**usePrevious**: Principal solução....

**useClientReact**: Principal solução....

# Hooks - part 1

Para entender um pouco melhor sobre este conceito vamos falar sobre estados e os 2 Hooks mais utilizados, sendo eles:

**useState**: Serve para controlar os estados de nossa aplicação.

**useEffect**: Invocado assim que nosso componente é iniciado.

## O que são estados de nossa aplicação?

Estados são armazenadores de informação local ou seja onde armazenamos dados de nosso componente que podem variar desde um valor Booleano a um objeto complexo.

Sendo assim os valores de nosso estado encerra o ciclo de vida quando "destruído" ou seja quando o componente não está sendo renderizado.



```
import React, {useState, useEffect} from 'react';
// importando o react e desestruturando os Hooks useState e useEffect
const PartOne: React.FC = () => {
  const [name, setName] = useState<string>();
  // criando um estado que recebe um string simples
  useEffect(() => {
    setName('Douglas')
  }, [])
  // iniciando o nosso componente com um valor em nosso estado
  return(
    <div>
      <h3>Olá, {name}.</h3>
      <input type="text" placeholder="Informe seu nome"
        value={name} onChange={e => setName( e.target.value )}
      />
      /* Recebendo evento de change e atualizando o estado */
    </div>
  );
}

export default PartOne;
// exportando o nosso componente
```

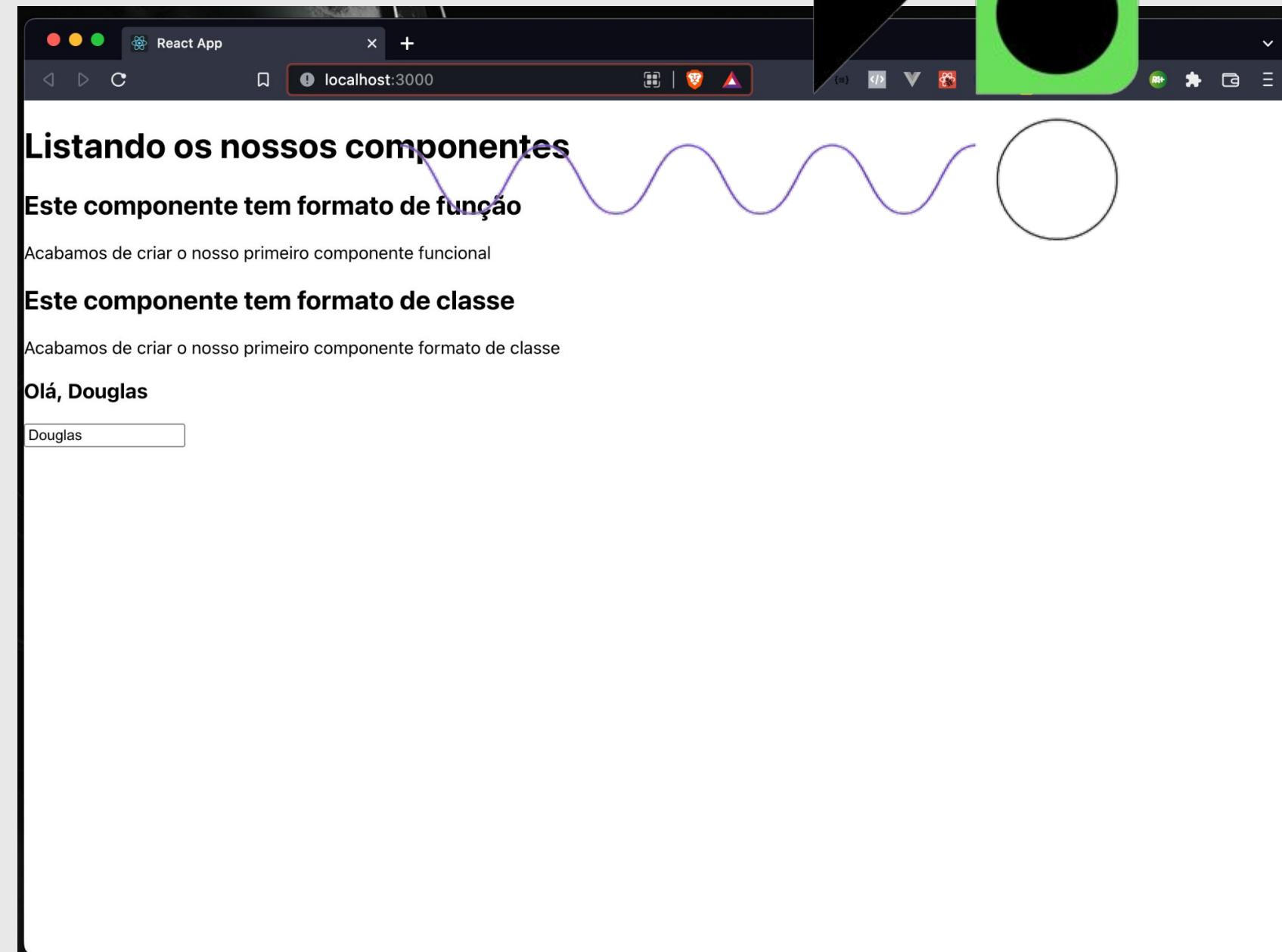
# Hooks - part 2

Ao lado podemos observar o resultado de nosso código após realizar as importações em nosso arquivo App.tsx. Lembrando assuntos importantes principalmente relacionado a sintaxe .

Quando invocamos o nosso useState, nós associamos ele a uma constante e declaramos um Array sendo o primeiro do índice o que recebe o valor e o segundo que seta um novo valor, por isto a seguinte nomenclatura.

```
const [name, setName] = useState();
```

Em nosso useEffect recebe um arrow function em um Array em seguida dentro deste Array devemos informar se queremos que observe um estado ou se omitimos em nosso caso omitimos [].



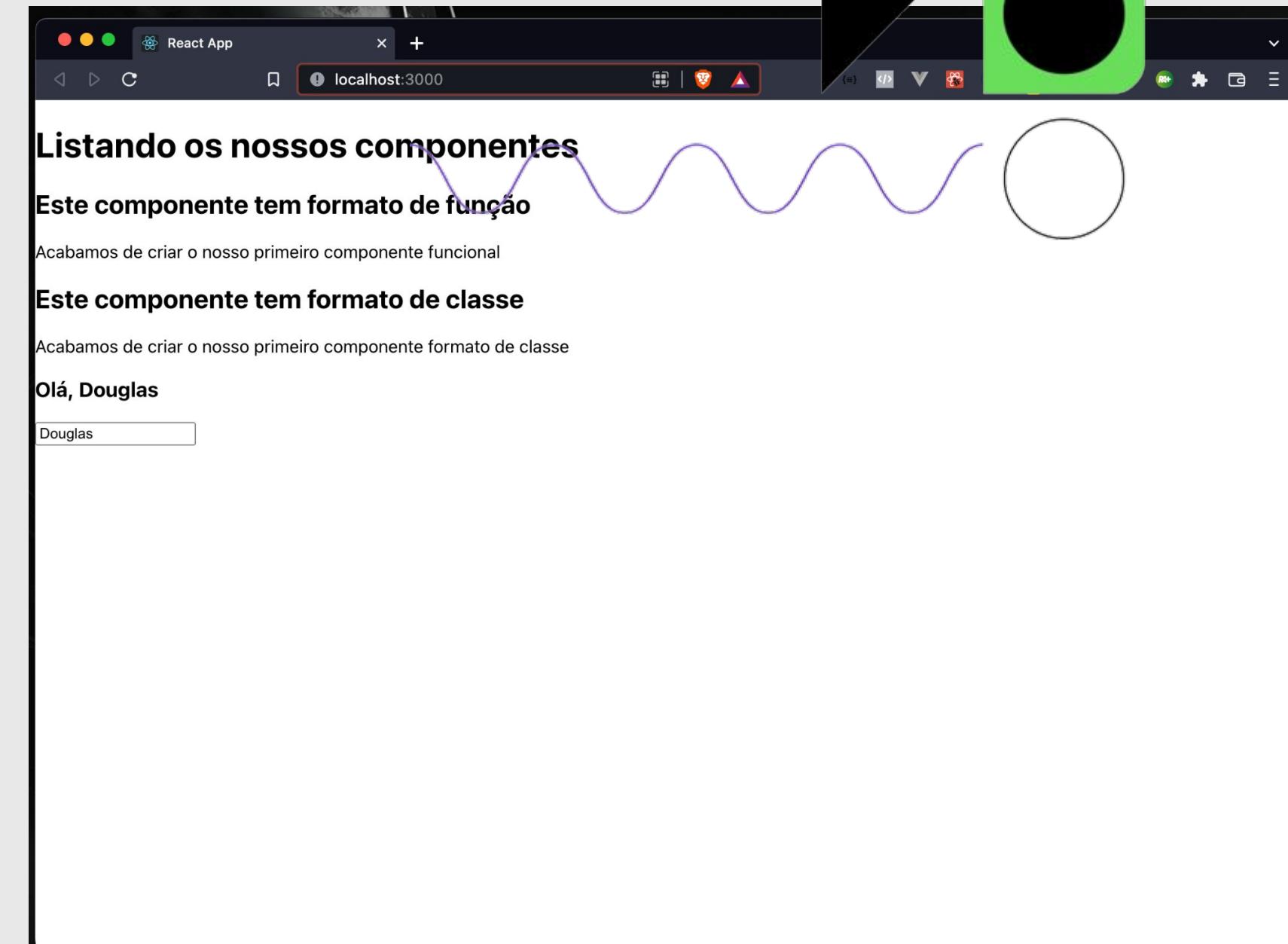
#PraCegoVer: Navegador aberto com nosso projeto rodando em ambiente de homologação.

# Hooks - part 3

Desmistificando, em nosso exercício criamos um componente que recebe uma string simples, em seguida o nosso Hook gerenciador de ciclo de vida (useEffect) é invocado e nosso estado recebe um valor "Douglas" no caso. Porém atribuímos um setName a um evento de nosso input onde o novo valor é associado ao nosso estado e assim que acionado o nosso estado passa a ter um novo valor imediatamente.

E se por algum motivo nossa aplicação for recarregada, o que acontece?

Ela passa a ter o valor inicial informado em nossa aplicação.



#PraCegoVer: Navegador aberto com nosso projeto rodando em ambiente de homologação.

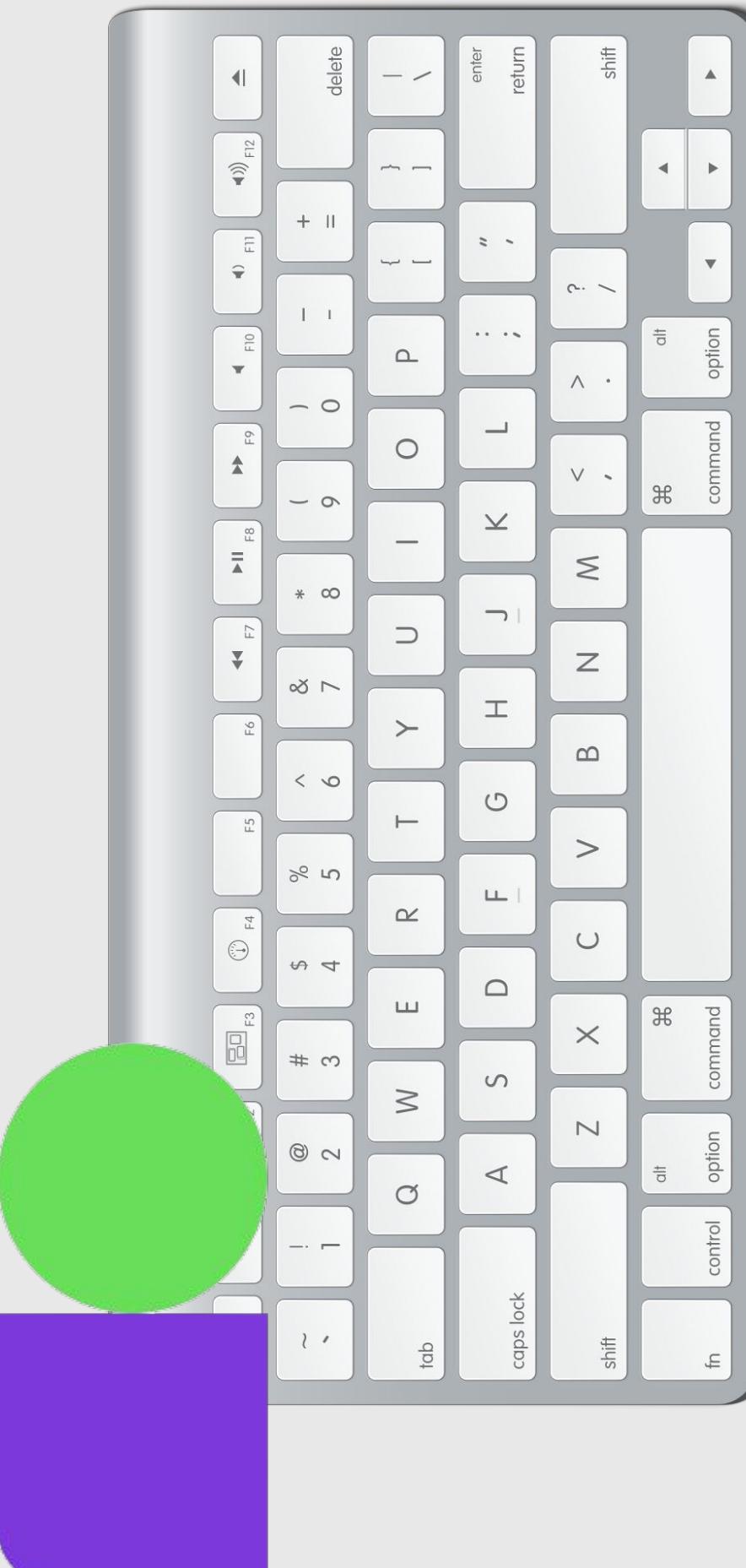
# Conhecendo os Hooks

Neste módulo aprendemos sobre os *Hooks* e seus super poderes.

Agora é colocar em prática todos estes conceitos e entendendo mais a fundo qual a proposta de cada um deles fica mais fácil ter a tomada de decisão de qual *Hook* usar e qual o melhor para implementar mediante a necessidade do desenvolvimento.

E é claro que estou disponibilizando o commit desta aula para auxiliar com o aprendizado, clique [aqui](#).

Para saber mais a fundo sobre os hooks clique [aqui](#).



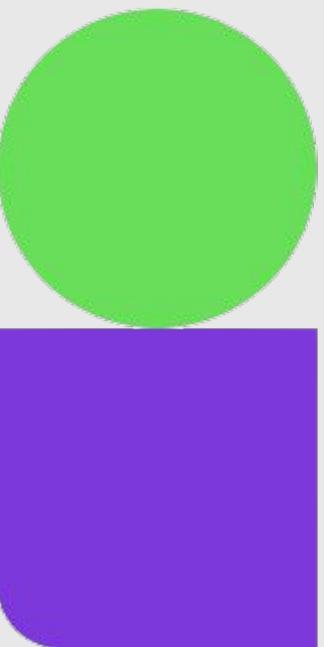


# Trabalhando com Rotas

*Criando página e definido rotas.*

#PraCegoVer: Dois notebooks  
o primeiro apresentado o teclado  
o segundo apresentando o  
monitor.

# Introdução



Neste módulo vamos aprender a trabalhar com rotas e para que serve as rotas?

Para tratar as página de nossa aplicação, e maneira muito simples, vamos criar uma página e criar um botão onde as rotas serão gerenciadas.

E para iniciar o nosso aprendizado teremos que instalar algumas dependências adicionais e atente-se ao gerenciador de dependências utilizado, em caso vou utilizar o *Yarn* por preferência, porém você pode usar o *NPM* que já vem instalado por padrão nas dependências do *NodeJS*.

# Instalando dependências.



Para iniciar as nossas rotas vamos precisar instalar as seguintes dependências:

react-router-dom (como dependência de produção)

@types/react-router-dom (como dependência de desenvolvimento)

```
→ via • v16.13.0 yarn add react-router-dom  
→ via • v16.13.0 yarn add @types/react-router-dom -D
```

Após a instalação das dependências mencionadas acima, devemos iniciar as configurações do nosso arquivo de rotas dentro do projeto.

Agora dentro de src vamos criar o nosso arquivo de rotas src/routes.tsx

```
Success! Uninstalled packages  
/Users/douglasmorais/Desktop/gama-react-ts/src/App.tsx  
└─ Done in 3.01s.  
  
douglasmorais in gama-react-ts on ✘ main [!] took 4s  
→ via • v16.13.0 yarn add @types/react-router-dom -D  
yarn add v1.22.15  
warning ../../package.json: No license field  
[1/4] 🔎 Resolving packages...  
[2/4] 🚀 Fetching packages...  
[3/4] ⚡ Linking dependencies...  
warning " > @testing-library/user-event@13.5.0" has unmet peer dependency "@testing-libr  
warning "react-scripts > tailwindcss@3.0.23" has unmet peer dependency "autoprefixer@^10  
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unm  
5".  
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unm  
x@^7.14.9".  
[4/4] ↗ Building fresh packages...  
success Saved lockfile.  
success Saved 2 new dependencies.  
info Direct dependencies  
└ @types/react-router-dom@5.3.3  
info All dependencies  
└ @types/react-router-dom@5.3.3  
└ @types/react-router@5.1.18  
└─ Done in 3.10s.  
douglasmorais in gama-react-ts on ✘ main [!] took 3s  
→ via • v16.13.0
```

#PraCegoVer: Terminal após a instalação das dependências de rotas instaladas.

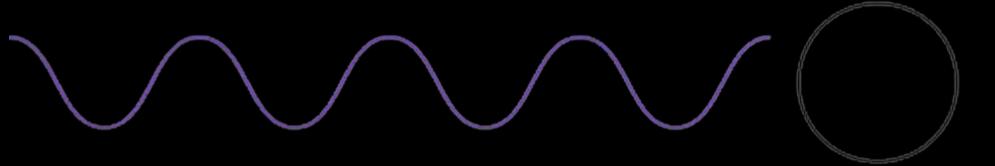
# Nosso arquivo de rotas

```
import React from 'react';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

import Home from './views/Home'
import Contato from './views/Contato'

const Router: React.FC = () => {
  return (
    <BrowserRouter>
      <div>
        <h2>Aqui está o projeto</h2>
        <Link to="/">Home</Link>
        <Link to="/contato">Contato</Link>
      </div>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/contato" element={<Contato />} />
      </Routes>
    </BrowserRouter>
  );
}

export default Router;
```



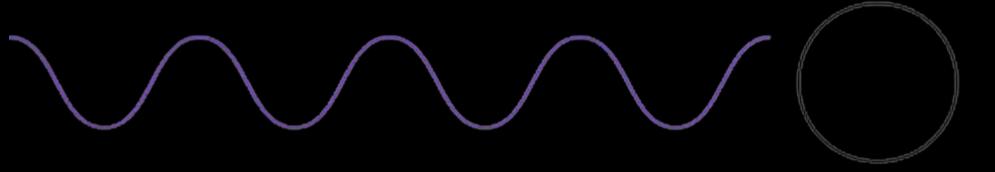
# Nosso arquivo App.tsx

```
// importação do React
import React from 'react';

// importação de nossas rotas
import Router from './routes'

const App: React.FC = () => {
  return <Router />;
}

export default App;
// exportação
```



# Nosso arquivo Home.tsx

```
import React from 'react';

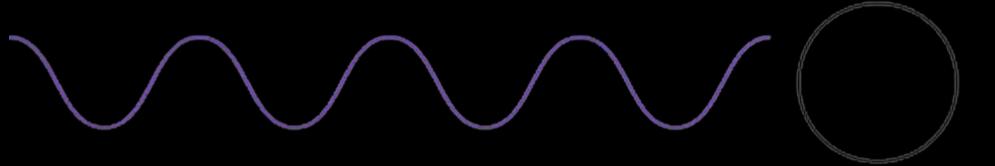
import FunctionalComponent from '../../components/FunctionComponents';
import ClassComponents from '../../components/ClassComponents';

import PartOne from '../../components/Hooks/PartOne';
import ButtonProps from '../../components/Props/Component';

const Home: React.FC = () => {
  return(
    <div>
      <h1>Listando os nossos componentes</h1>
      <FunctionalComponent/>
      <ClassComponents/>
      {/* importando adicionando ao código os componentes */}
      <PartOne />

      <ButtonProps name="Botão 1" />
      <ButtonProps name="Botão 2" />
      {/* invocando os componentes */}
    </div>
  );
}

export default Home;
```



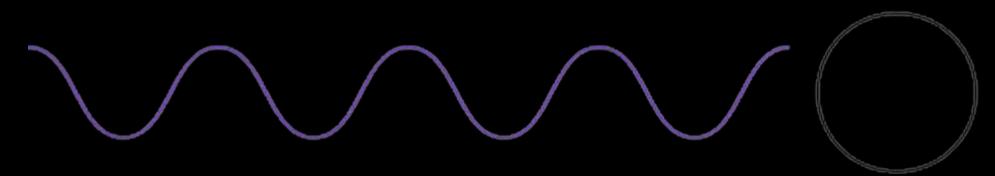
# Nosso arquivo Contato.tsx

```
import React from 'react';

// import { Container } from './styles';

const Contato: React.FC = () => {
  return (
    <form>
      <input type="text" placeholder="Informe seu nome"/>
      <input type="text" placeholder="Informe seu email"/>
      <input type="text" placeholder="Informe seu telefone"/>
      <input type="submit" value="Enviar" />
    </form>
  );
}

export default Contato;
```

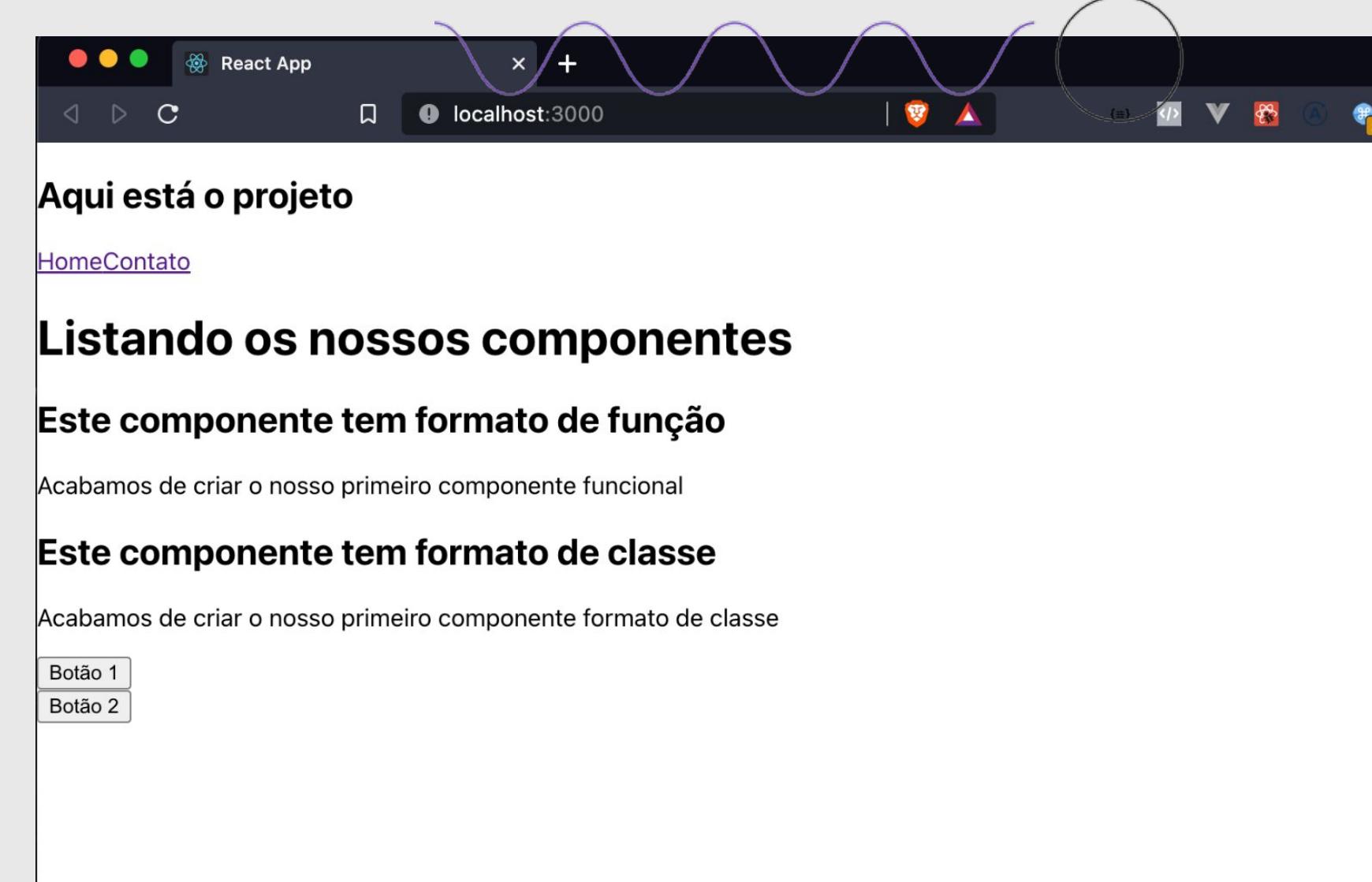


# Resultado de final

Toda a estrutura e organização dos nossos arquivos estão comitados no GitHub para simplificar seu processo de aprendizado.

Atente-se a organização de arquivos e principalmente a importação de cada um deles é de extrema importância para o funcionamento do seu projeto.

Para acompanhar as etapas de organização do projeto.



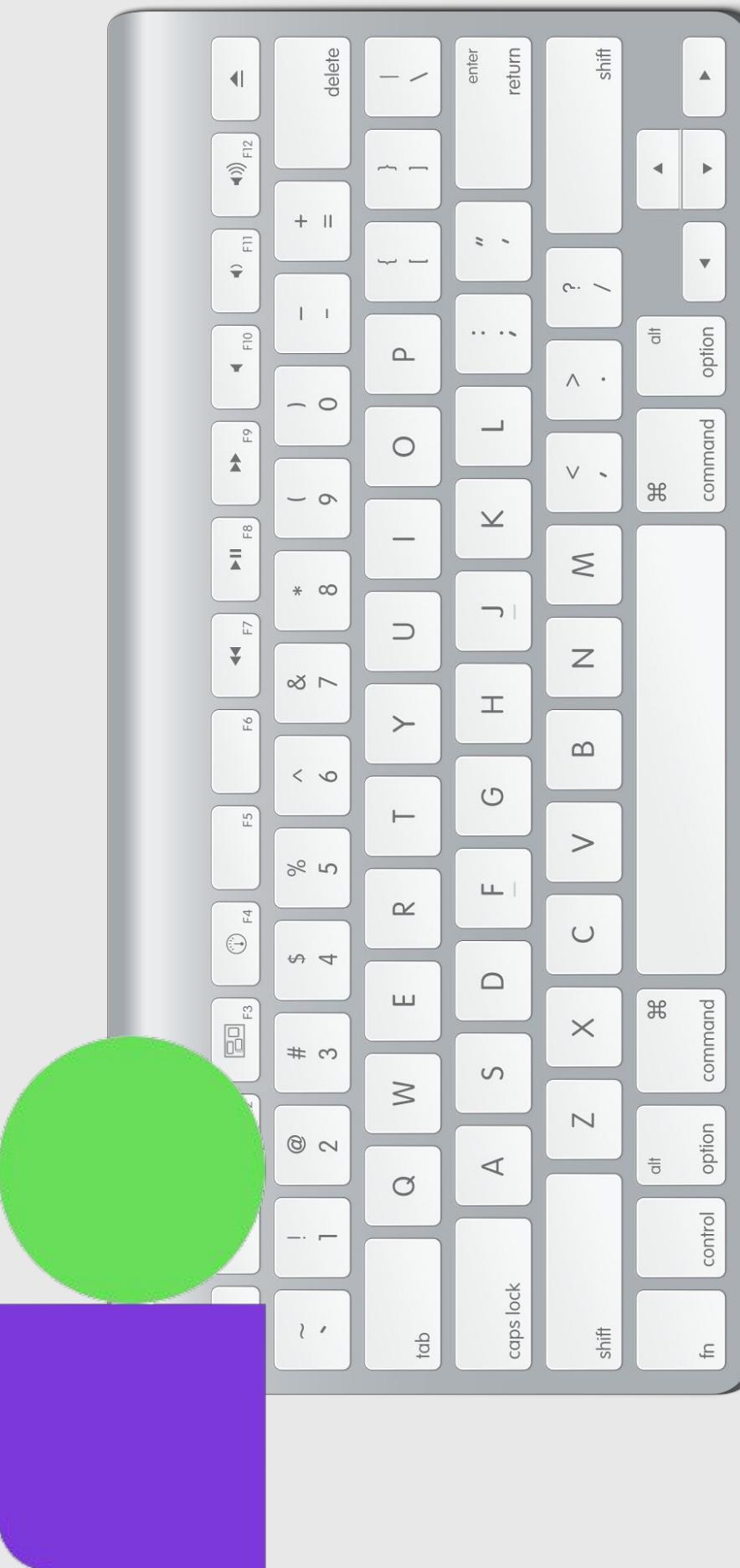
#PraCegoVer: Nosso projeto em ambiente de desenvolvimento.

# Trabalhando com Rotas

Neste módulo aprendemos as práticas mais atuais para trabalhar com rotas dentro do React.

Este módulo abordou a maneira atualizada de tratar rotas e componentes, a únicas ocorreram na versão 17.x.x do React e na versão 6.2.2 do React Router Dom, para mais informações dos módulos clique [aqui](#).

Para acompanhar o seu projeto clique [aqui](#) e veja commit no github.



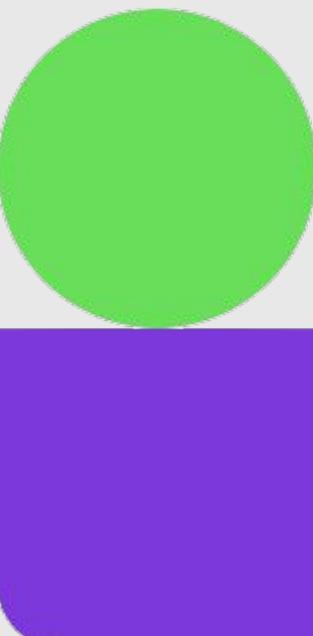


# Trabalhando com Props

#PraCegoVer: Dois notebooks  
o primeiro apresentado o teclado  
o segundo apresentando o  
monitor.

*Componentes reaproveitamento de  
código e suas propriedades.*

# Introdução



Neste módulo vamos aprender a trabalhar com *Props* e o que são?

Basicamente uma maneira de passar valor para componentes ou torná-los dinâmicos, como assim?

Podemos criar um componente reutilizável em nosso projeto onde um determinado texto ou cores podem ser alterados no ato de importação do mesmo para o nosso projeto.

A seguir teremos um exemplo muito interessante fazendo uso desta feature.

# Class component

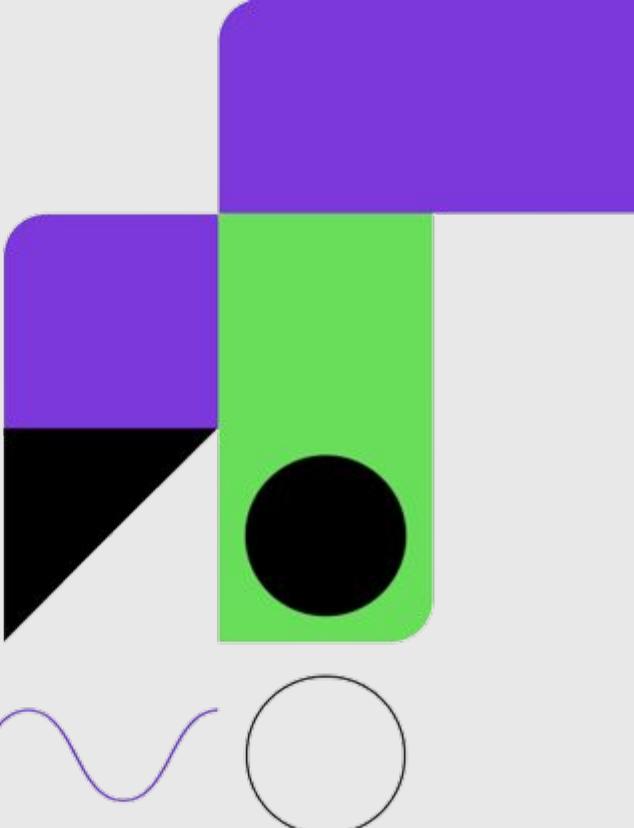
Para este exercício nós vamos criar um botão e alterar o nome dele no ato da importação ou seja o texto apresentado na renderização será definido por meio de uma Props.

Para iniciar vamos criar em nossa estrutura do projeto o seguinte arquivo:

*src/components/Props/Components/index.tsx*

Dentro deste arquivo nós vamos criar o nosso componente e importar dentro do nosso arquivo App.tsx.

A seguir a estrutura de nosso arquivo.



```
index.tsx -- gama-react-ts
import React from 'react';
import './index.css';

const Button: React.FC<IButtonProps> = ({name}: IButtonProps) => {
  // estamos informando nosso componente que recebemos uma props
  // mas no fim das contas o que é uma props?
  // uma propriedade passado de um componente para o outro.
  return (
    <div>
      <button onClick={() => alert(name)}>{name}</button>
    </div>
  );
}

export default Button;
```

#PraCegoVer: VSCode com arquivo *src/components/Props/Components/index.tsx* aberto.

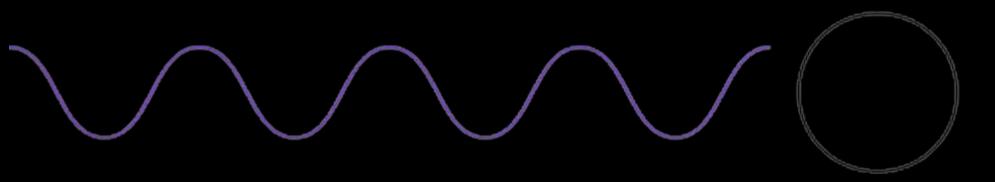
# Nosso arquivo

```
import React from 'react';
// importação do React

interface IButtonProps{
  name: string;
}

const Button: React.FC<IButtonProps> = ({name} : IButtonProps) => {
  // estamos informando nosso componente que recebemos uma props
  // mas no fim das contas o que é uma props?
  // uma propriedade passado de um componente para o outro.
  return (
    <div>
      <button onClick={() => alert(name)}>{name}</button>
    </div>
  );
}

export default Button;
```

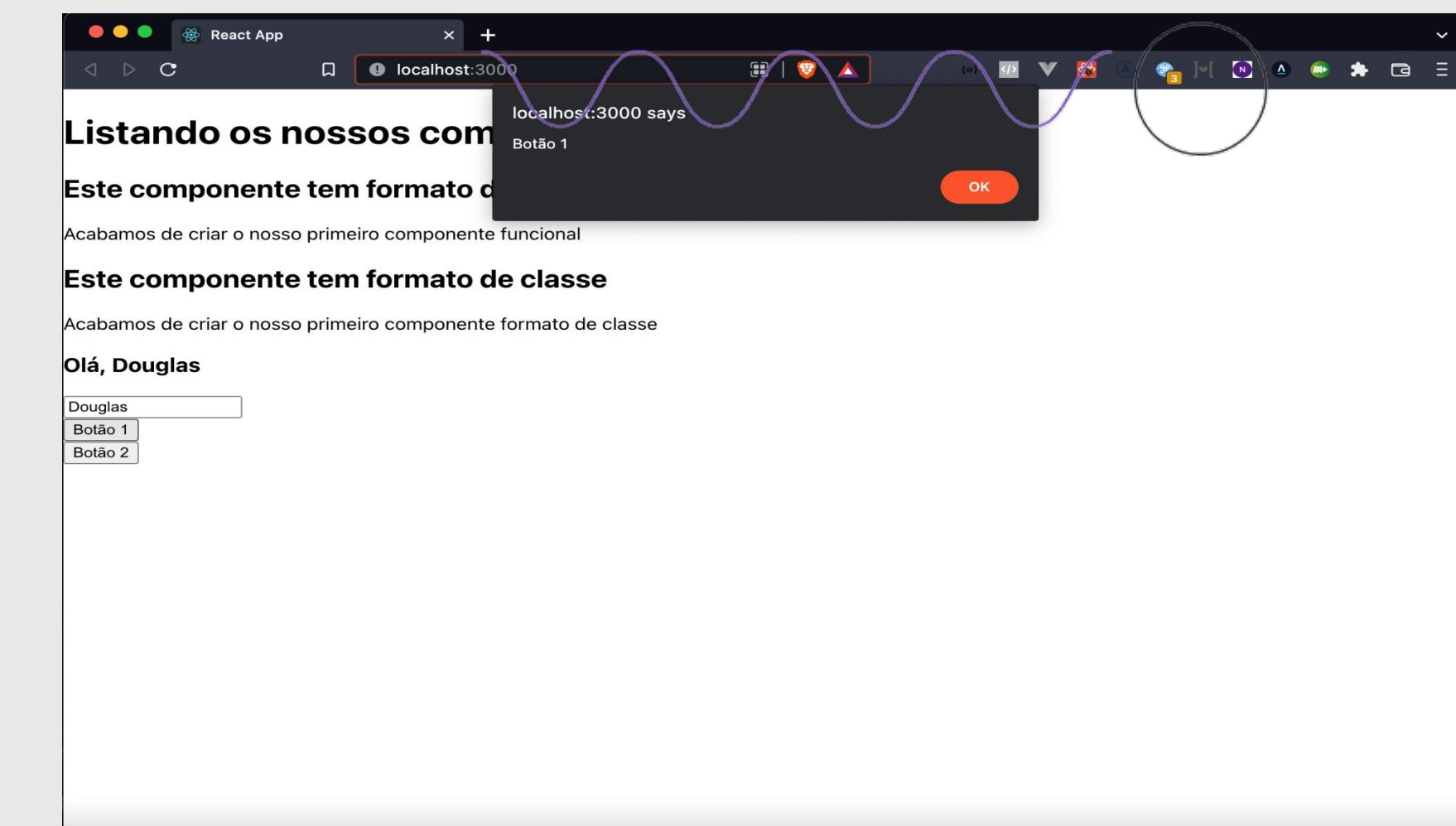


# Resultado

Por fim o resultado de nosso arquivo a seguir:

Em nosso arquivo App.tsx nós adicionamos as seguintes linhas:

```
import ButtonProps from './components/Props/Component'  
// importando o nosso componente  
  
...  
  
<ButtonProps name="Botão 1" />  
<ButtonProps name="Botão 2" />  
/* invocando os componentes */
```



Na próxima página teremos o arquivo completo, lembrando que as linhas mencionadas acima são para serem adicionadas.

#PraCegoVer: VSCode com arquivo src/components/Props/Components/index.tsx aberto.

# Nosso arquivo App.tsx

```
// importação do React
import React from 'react';

// importação dos componentes
import FunctionalComponent from './components/FunctionComponents';
import ClassComponents from './components/ClassComponents';

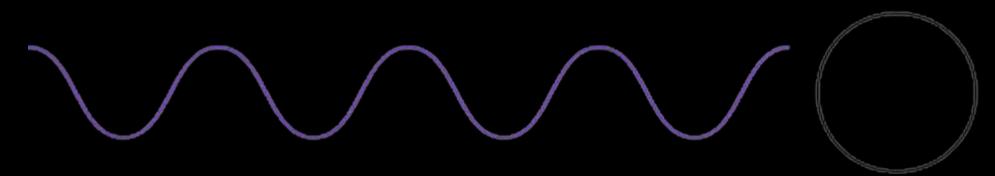
import PartOne from './components/Hooks/PartOne'

import ButtonProps from './components/Props/Component'
// importando o nosso componente

// componente no formato de função
const App: React.FC = () => {
  return(
    <div>
      <h1>Listando os nossos componentes</h1>
      <FunctionalComponent/>
      <ClassComponents/>
      {/* importando adicionando ao código os componentes */}
      <PartOne />

      <ButtonProps name="Botão 1" />
      <ButtonProps name="Botão 2" />
      {/* invocando os componentes */}
    </div>
  );
}

export default App;
// exportação
```

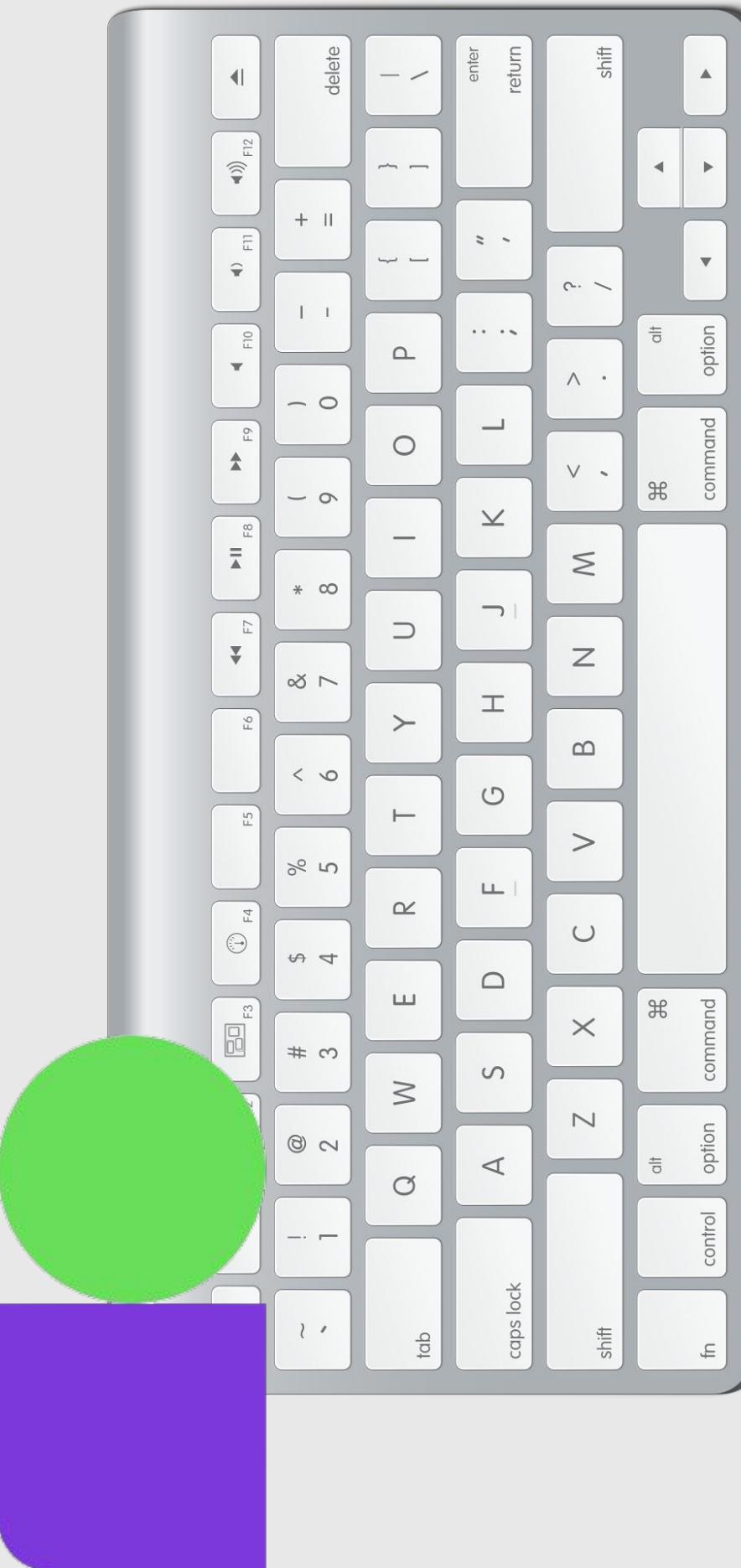


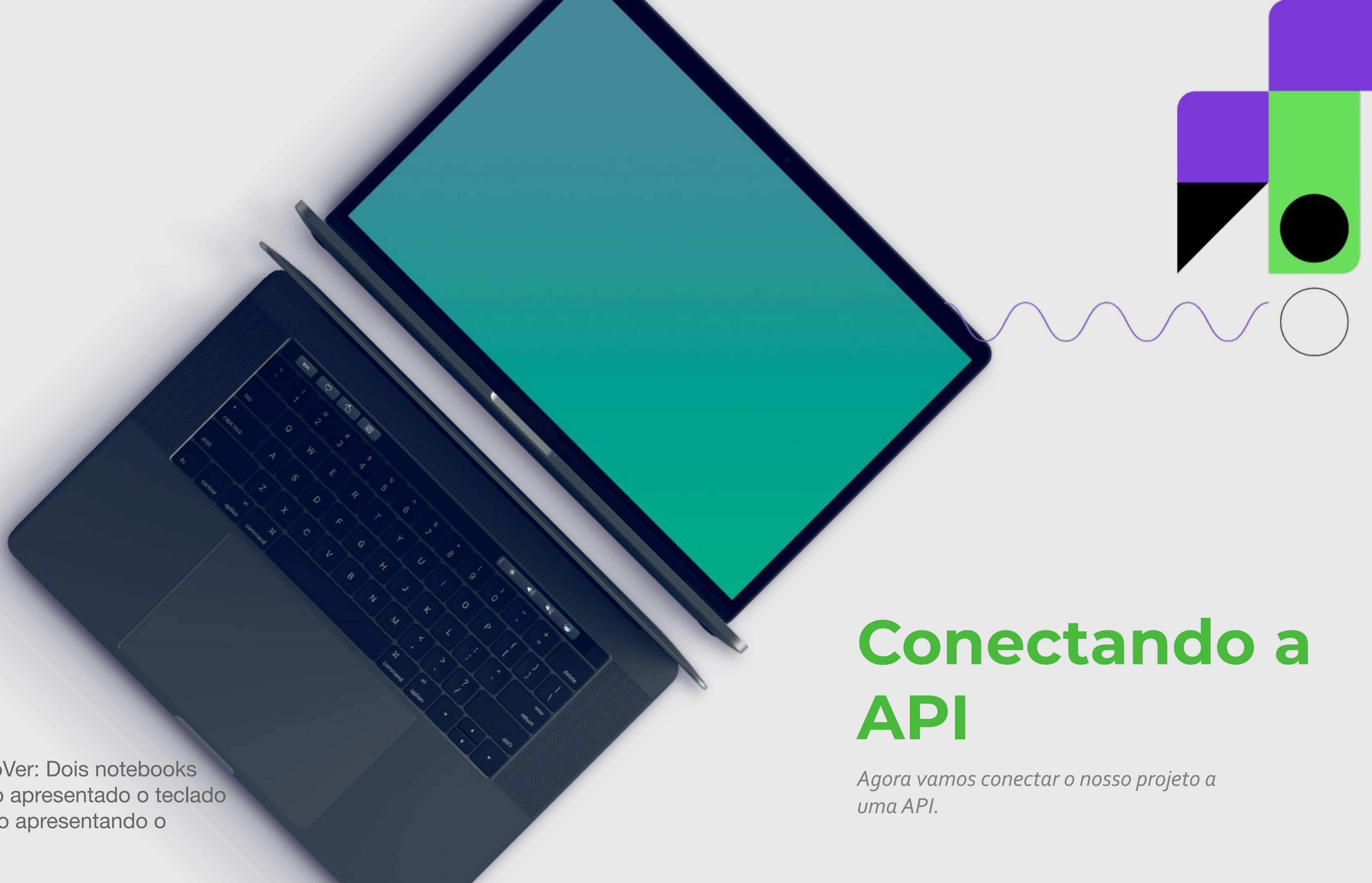
# Trabalhando com props

Por fim tivemos uma breve introdução às Props com React e apesar de ser um exemplo simples, temos uma funcionalidade muito importante para projetos, sua aplicabilidade reutilização de componentes ou até mesmo uso em um design systems e afins.

Este exemplo foi um abordagem superficial sobre o real potencial das Props e contexto do projeto.

Como boa prática segue link com o commit do projeto, clique [aqui](#).



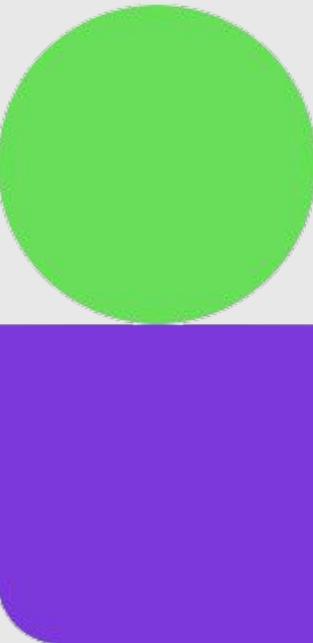


# Conectando a API

*Agora vamos conectar o nosso projeto a uma API.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Introdução



Neste módulo vamos conectar a nossa a uma API e encaminhar dados para ela através do método *POST* e para este trecho, vamos instalar uma dependência chamada AXIOS. Ela vai nos auxiliar com *Client Data Fetching*.

Para os próximos passo vamos utilizar um serviço chamado Hook.site e o que ele é? Um serviço de webhook dinâmico que gera uma URL única para nós e podemos enviar dados para ele e validar se está tudo okay com nossa aplicação.

Agora vamos lá, nosso módulo está quase no final.

# Instalando dependências.



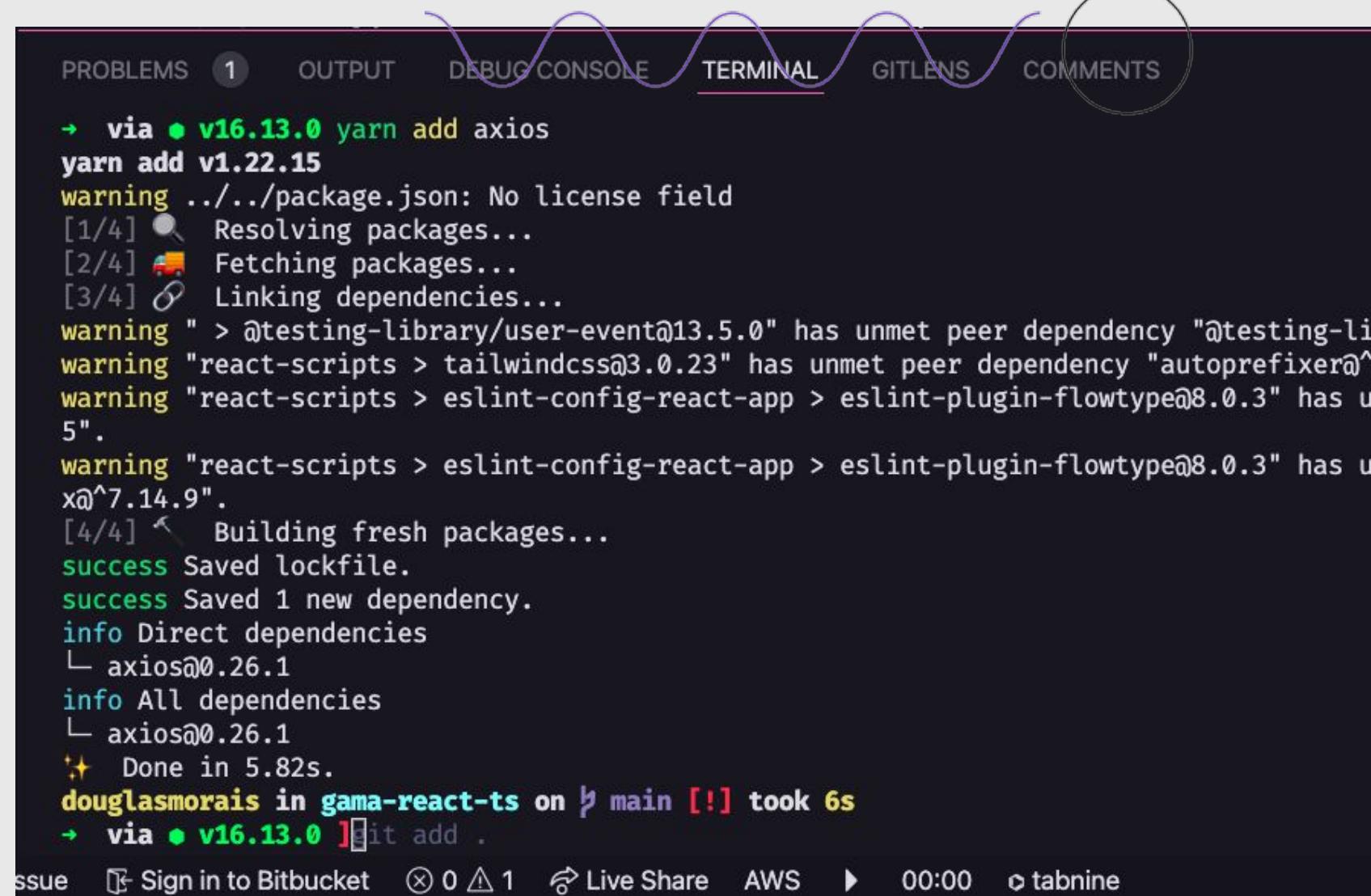
Para iniciar o nosso service de api, vamos instalar o axios em nosso projeto utilizando o comando abaixo.

```
→ via • v16.13.0 yarn add axios
```

Após a instalação em nosso projeto, a tela ao lado estará visível com a mensagem de sucesso.

A seguir devemos acessar o seguinte site (webhook.site), clique [aqui](#).

Agora devemos criar uma pasta de services dentro de nosso projeto e centralizar a importação do axios dentro dos mesmos.



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL GITLENS COMMENTS

→ via • v16.13.0 yarn add axios
yarn add v1.22.15
warning .../package.json: No license field
[1/4] ⚡ Resolving packages...
[2/4] 🚛 Fetching packages...
[3/4] ⚡ Linking dependencies...
warning " > @testing-library/user-event@13.5.0" has unmet peer dependency "@testing-lib
warning "react-scripts > tailwindcss@3.0.23" has unmet peer dependency "autoprefixer@^1
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has un
5".
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has un
x@^7.14.9".
[4/4] ⚡ Building fresh packages...
success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└─ axios@0.26.1
info All dependencies
└─ axios@0.26.1
✨ Done in 5.82s.
douglasmorais in gama-react-ts on ✖ main [!] took 6s
→ via • v16.13.0 ]git add .
```

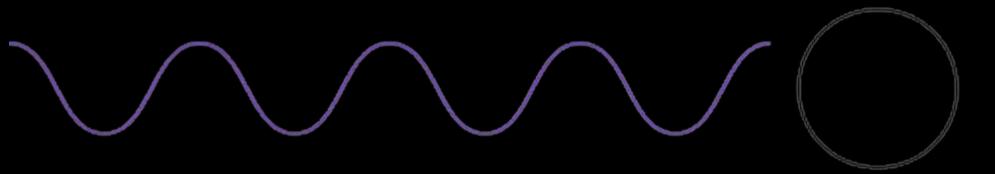
#PraCegoVer: VSCode com a dependência do AXIOS instalada.

# Nosso arquivo api.ts

```
import axios from 'axios';

const api = axios.create({
  baseURL: "https://webhook.site/4c639737-c1c9-488d-a085-013d2a26e614"
})

export default api
```

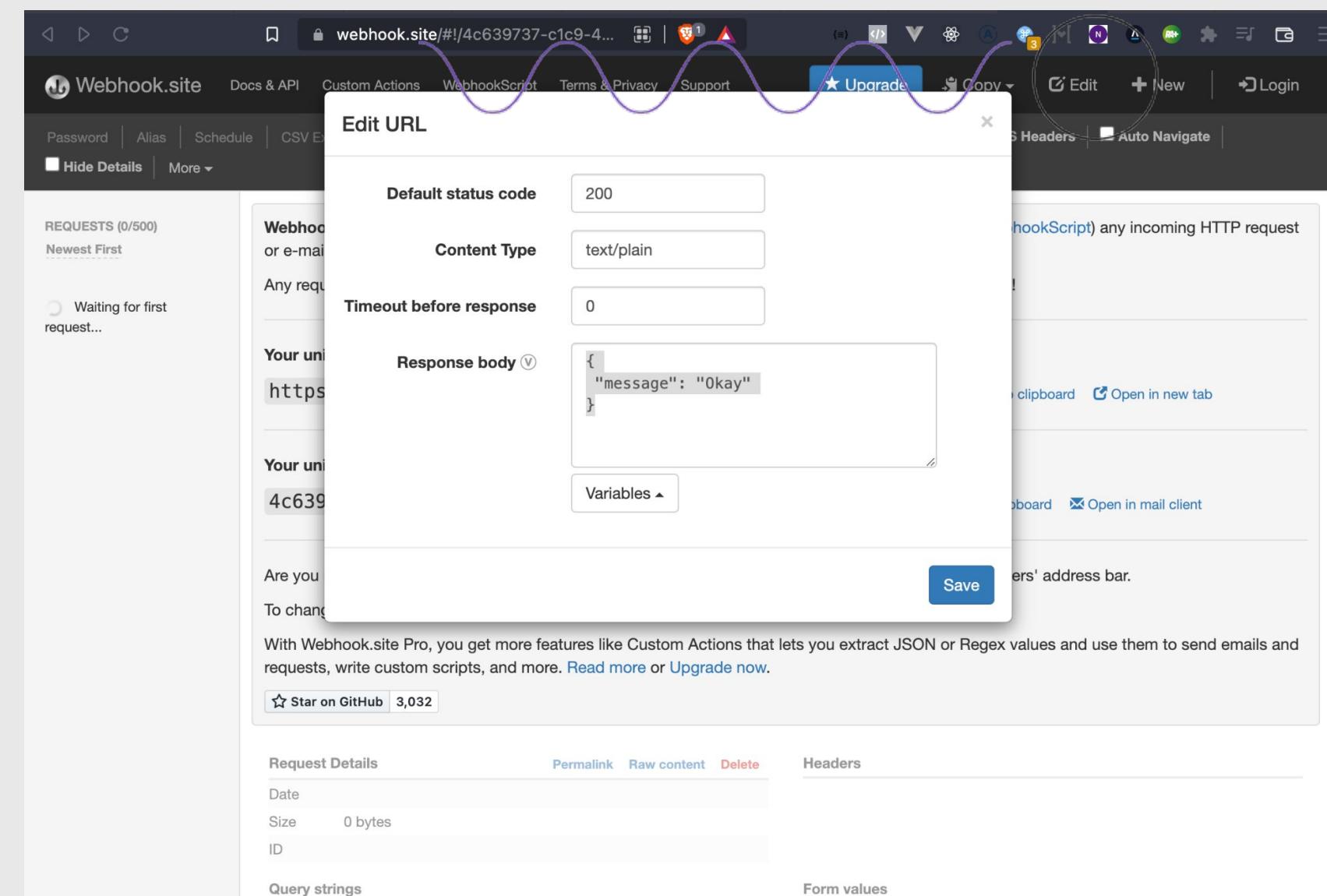


# Importando a service

Agora devemos importar nossa service dentro do nosso arquivo de contato, que criamos junto a nossas rotas.

Iremos até o webhook.site e lá devemos selecionar a caixa CORS Headers na parte superior da página em seguida clicando em edit informar no formato JSON a seguinte mensagem:

```
{  
  "message": "Okay"  
}
```



#PraCegoVer: Navegador com o webhook.site aberto.

# Nosso arquivo Contato/index.ts

```
import React, {useState, FormEvent, useCallback} from 'react';
import api from '../..//services/api'

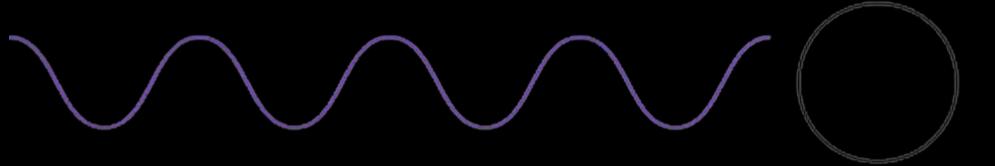
interface IDataContent{
  name: string;
  email: string;
  phone: string;
}

const Contato: React.FC = () => {
  const [data, setData] = useState<IDataContent>({} as IDataContent);

  const postData = useCallback((e: FormEvent<HTMLFormElement>) => {
    e.preventDefault()
    api.post('', data).then(
      res => alert(res.data.message)
    )
  }, [data])

  return (
    <form onSubmit={postData}>
      <input
        type="text"
        placeholder="Informe seu nome"
        onChange={e => setData({ ... data, name: e.target.value})}
      />
      <input
        type="text"
        placeholder="Informe seu email"
        onChange={e => setData({ ... data, email: e.target.value})}
      />
      <input
        type="text"
        placeholder="Informe seu telefone"
        onChange={e => setData({ ... data, phone: e.target.value})}
      />
      <input
        type="submit"
        value="Enviar"
      />
    </form>
  );
}

export default Contato;
```

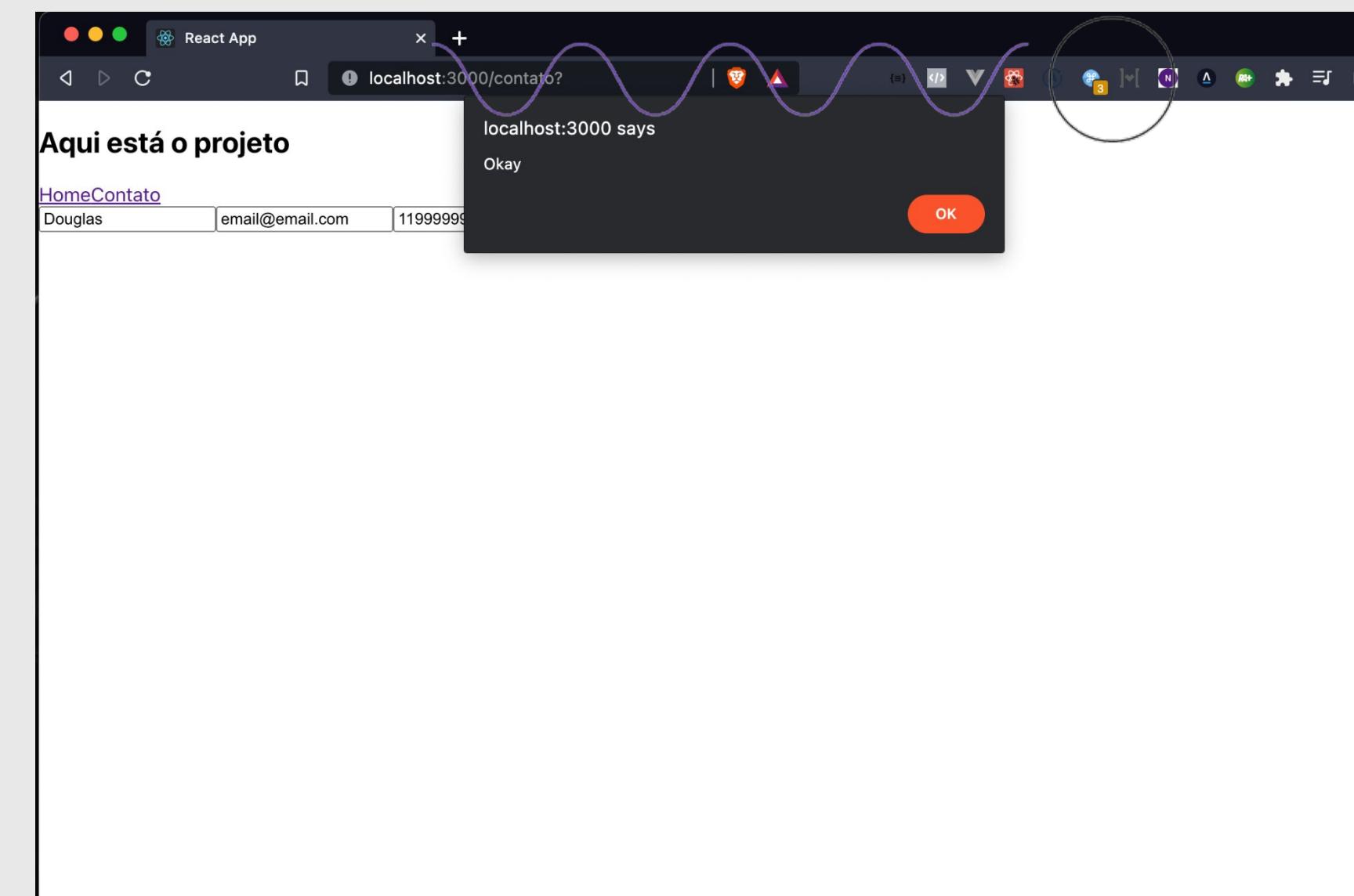


# Importando a service

Por fim temos o nossa primeira conexão a API finalizada.

Para conferir o resultado, visualiza o webhook.site e lá será possível enxergar os logs do objetos postados.

Agora falta pouco para finalizar o nosso projeto, nos passos a seguir vamos publicar em um serviço chamado vercel.



#PraCegoVer: Navegador com o webhook.site aberto.

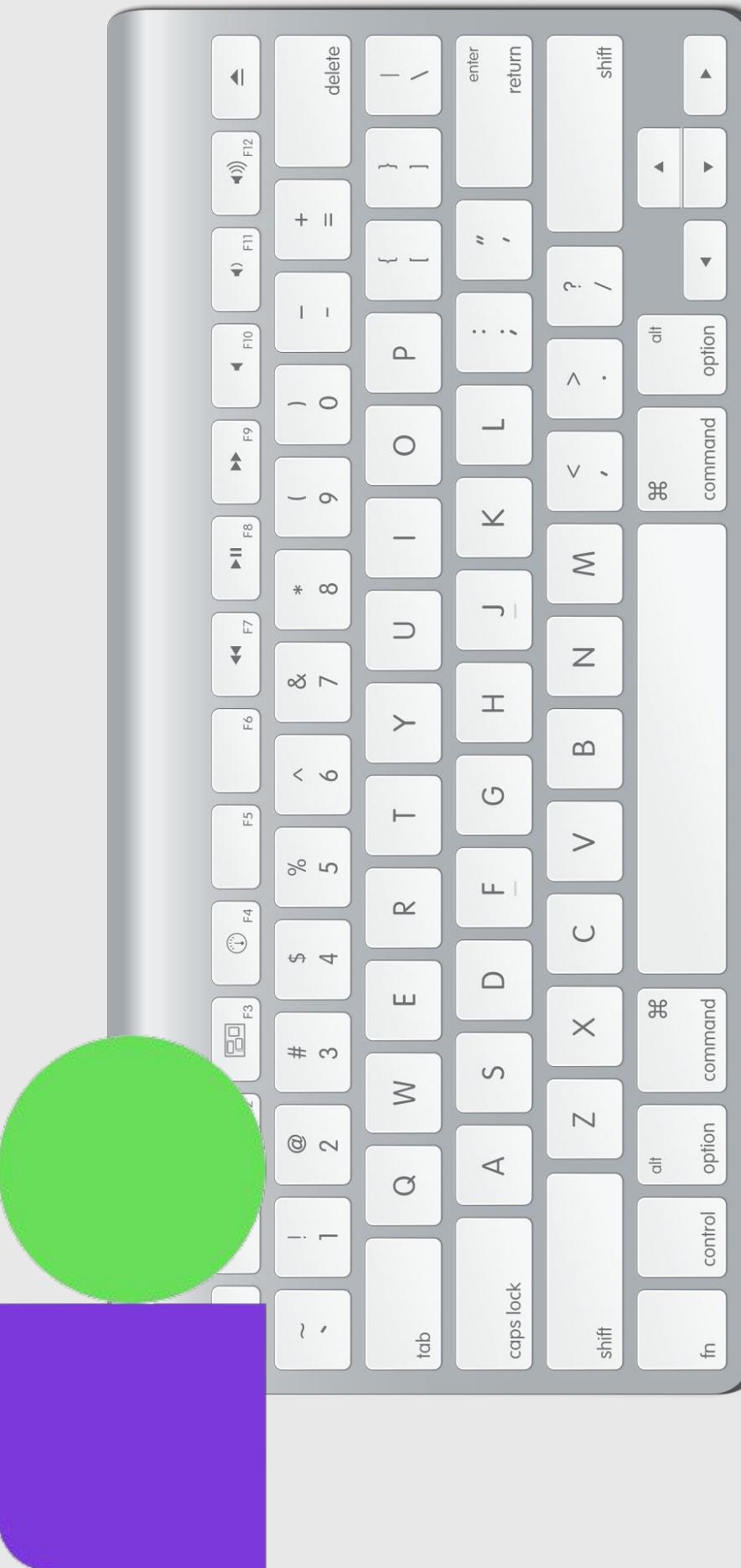
# Conectando a API

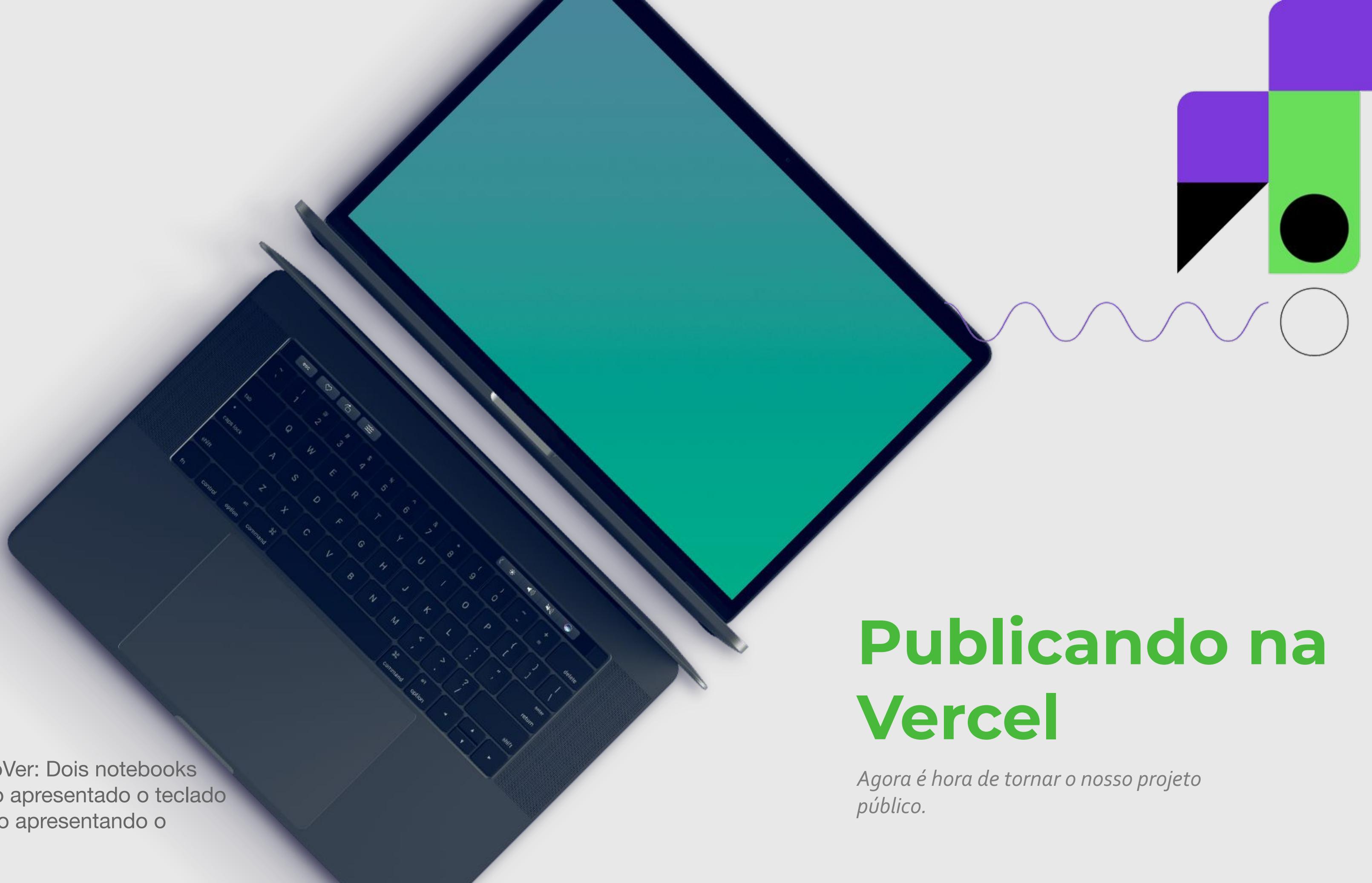
Por fim, neste módulo criamos nossa primeira conexão com API utilizando o AXIOS e as práticas mais recentes com Hooks do React.

Nossa jornada está quase no fim.

Para acessar o commit clique [aqui](#).

Até o próximo módulo.



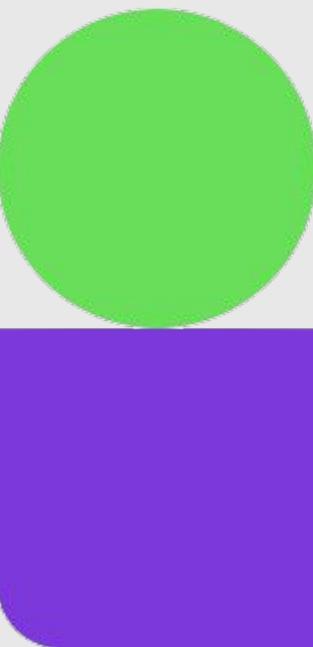


# Publicando na Vercel

*Agora é hora de tornar o nosso projeto  
público.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Introdução



Para este módulo vamos aprender a tornar o nosso projeto público e começar a popular o nosso portfólio, não se esqueça de compartilhar cada exercício no Linkedin com link do github e link de publicação.

Para este passo, vai ser necessário criar uma conta na Vercel, para acessar o site clique [aqui](#) e crie uma conta usando o seu Github.

Agora vamos colocar a mão na massa!

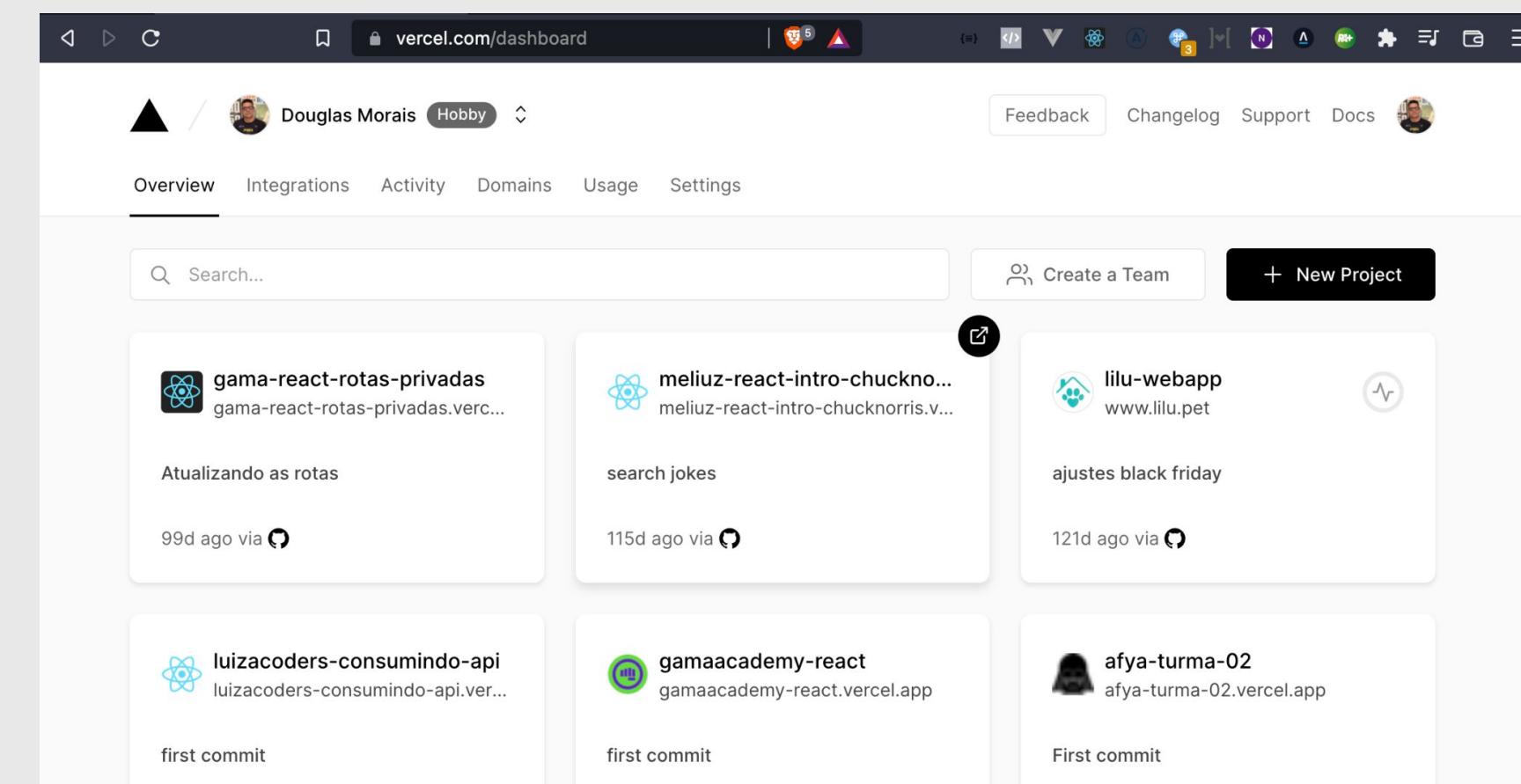
# Publicando na Vercel

Assim que logar no painel da Vercel, vamos precisar criar o nosso projeto, aqui em minha conta eu posso alguns, porém o processo é o mesmo para vocês.

Clique em **+ New Project** em seguida é necessário informar o repositório do Git no qual o projeto deve ser importado e clique em **Import**.

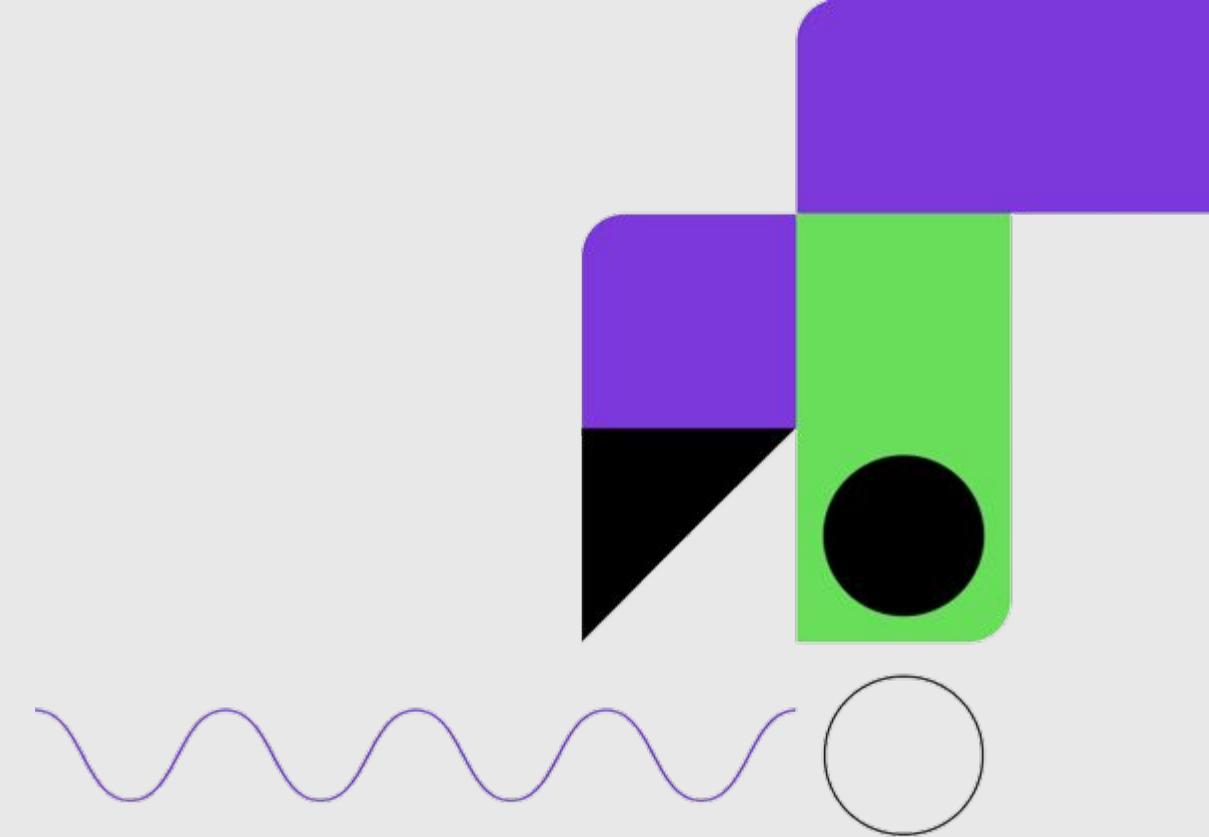
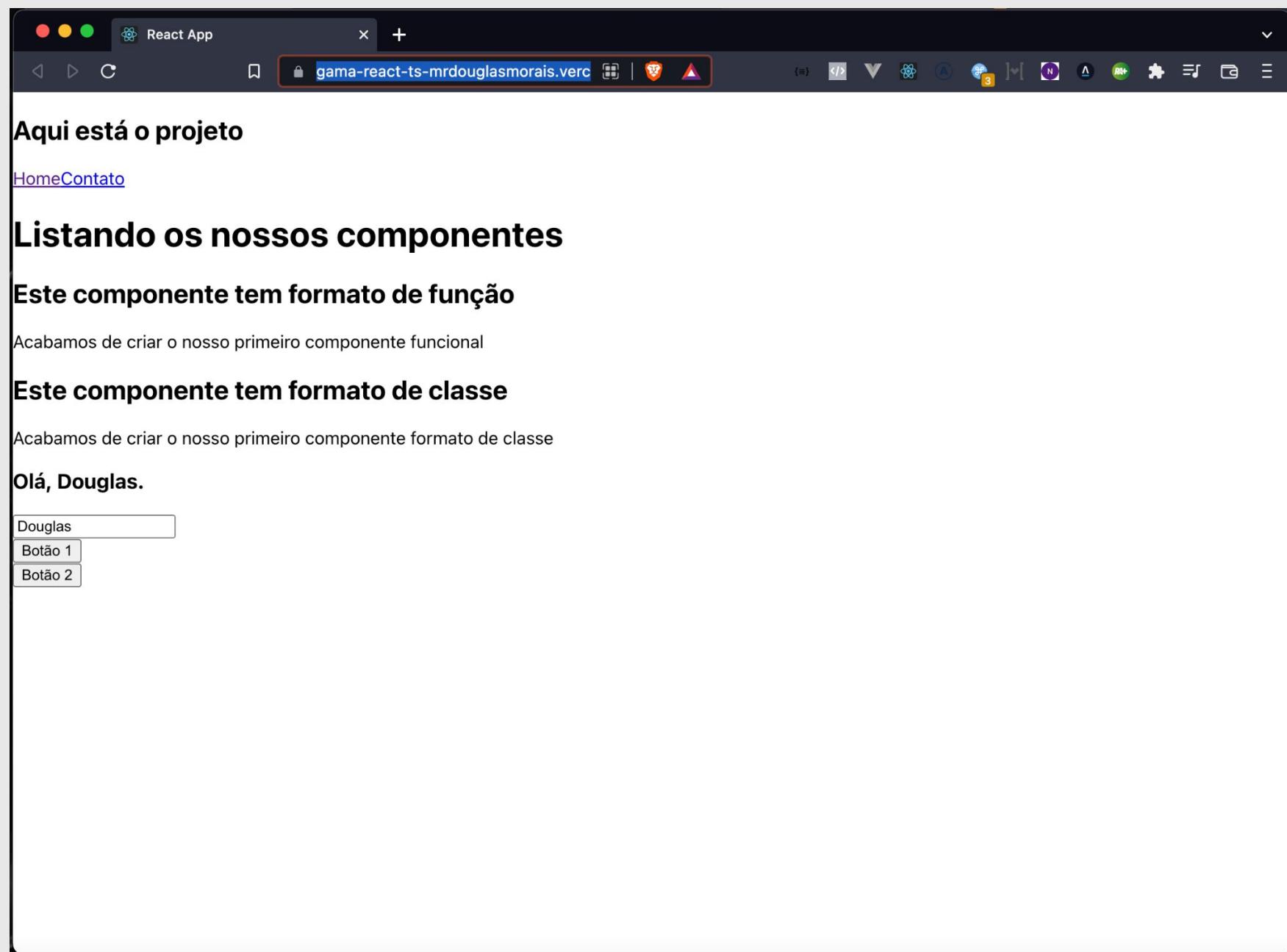
Não é necessário alterar nenhuma configuração nos passos a seguir, clique em **Deploy**.

Na sequência um processo de Build será iniciado em nosso projeto e por fim a plataforma te retorna com o link público do seu projeto.



#PraCegoVer: Navegador com o painel da Vercel aberto.

# Missão cumprida



Por fim temos o nosso projeto publicado.

Para acessar o link do meu projeto clique neste [aqui](#).

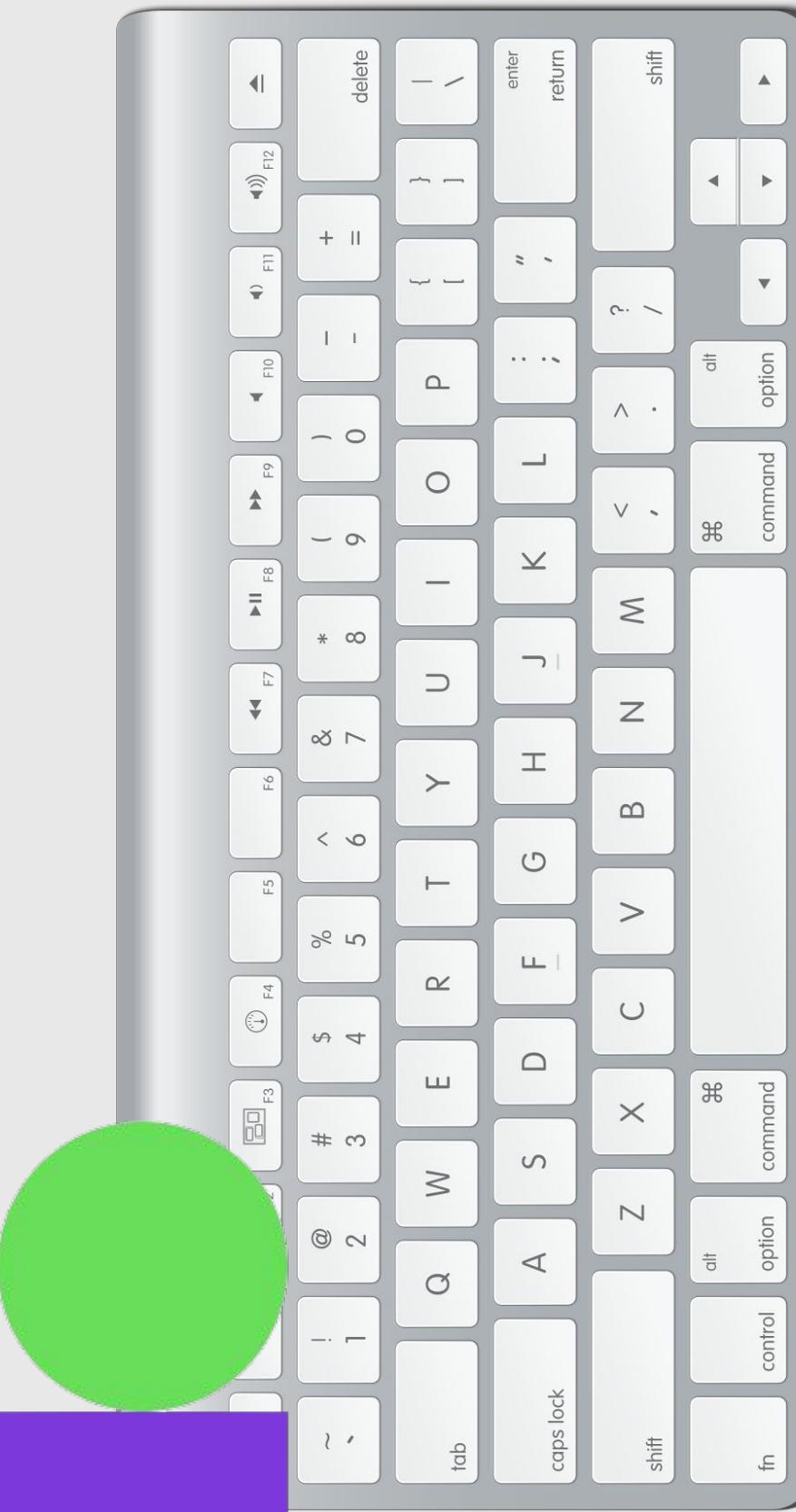
#PraCegoVer: Nosso projeto publicado na vercel.

# Publicando na Vercel

Neste módulo aprendemos a publicar nosso projeto na vercel vinculando nosso github a plataforma, assim gerando de maneira rápida uma CI (Continuous Integration) onde a cada commit realizado na branch main uma ação de deploy é iniciada.

E por fim temos o nosso primeiro projeto no ar.

E não se esqueçam, os estudos não param por aqui.



# Fechamento

## Sobre o nosso aprendizado.

Agora que entendemos conceitos e funcionalidades extrema importância do React é hora de avançar para o próximo nível.

Tenha clareza que conceitos de suma importância para o seu projeto estão em torno do conhecimento da linguagem que faz parte do contexto de uma solução, em nosso caso Javascript e o superset Typescript.

Siga-nos no Linkedin clique [aqui](#).



*E os estudos não param por aqui.*

*Aproveite ao máximo para praticar lembrando que...*

*"a repetição sem exaustão leva a perfeição" - Autor desconhecido.*

*Prof Douglas Moraes*

# Referência Bibliográfica

ReactJS - Documentação oficial.

Vercel - Documentação oficial.

