



React para iniciantes





<title>

AGENDA DO </> </> </> MÓDULO

</title>



Agenda do módulo



Anotações



{Como funcionam os componentes}

{Criando o primeiro componente}

{Propriedades}

{Criando o segundo componente}

{Introdução ao TypeScript}



React.js é a segunda tecnologia web mais comum usada por desenvolvedores profissionais.

Stack Overflow, 2022



<title>

MISSÃO DO MÓDULO

</title>

Criar seus primeiros
componentes em React.



{01.}

Como funcionam os componentes



+

+

+

+

+

+

A persistência é o melhor caminho do êxito.

Charles Chaplin



Os componentes em React se assemelham às funções JavaScript em alguns quesitos, pois aceitam parâmetros de entrada como propriedades, que são chamadas de “props”.

Além disso, suas principais vantagens também se assemelham às das funções. As principais são a reutilização, a abstração e o fato de evitar redundância no código.

Já a grande diferença é que em vez de termos apenas um arquivo para todo o script, utilizando React, temos um arquivo para cada componente, o que facilita a organização do projeto.

Quebre-o na Menor Unidade
Reaproveitável de código.

Dê um **MURro**
no seu
componente.



Os componentes podem ser criados com:

- › Function declaration
- › Function expression
- › Arrow function
- › Class

Dentre os tipos de componentes, a diferenciação principal entre eles se dá entre os class components, ou stateful components, e functional components, ou stateless components.

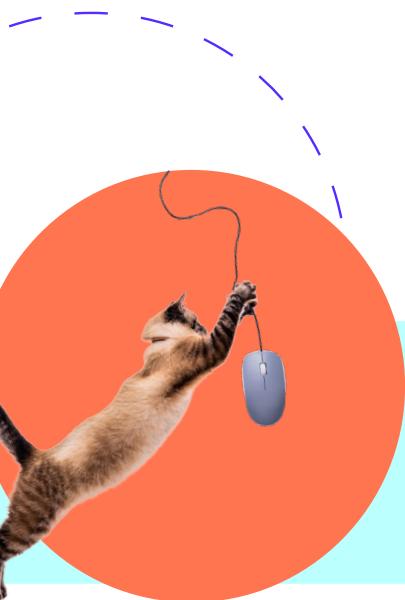
CLASS COMPONENTS (Stateful components)

- › A nomenclatura é declarada por meio de uma classe.
- › Exercem funções mais complexas.
- › Possuem um estado (state) e um ciclo de vida (life cycle).



FUNCTIONAL COMPONENTS (Stateless components)

- › Normalmente são declarados como uma função JavaScript.
- › Têm a função mais de display.
- › Os estados podem ser passados como props para os componentes filhos.



<title>

PULO DO GATO

</title>

- Class components caíram em desuso no React.
- Dê preferência para functional components com Hooks para gerenciar estado e ciclo de vida.

Os componentes React que trabalham com **states** têm um **ciclo de vida** definido dentro da aplicação.

Dizemos que os componentes que são montados em tela podem sofrer alteração durante seu uso e, após isso, são desmontados.

A cada passo do seu ciclo de vida, é possível chamar métodos interceptando sua renderização ou captando informações desse ciclo.

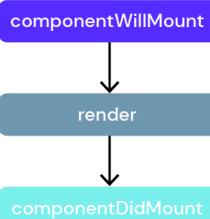


Ciclos de vida

Initialization (Inicialização)

setup props and state

Mounting (Montagem)



Updation (Atualização)

props

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render

states

shouldComponentUpdate

true --> componentWillUpdate

false

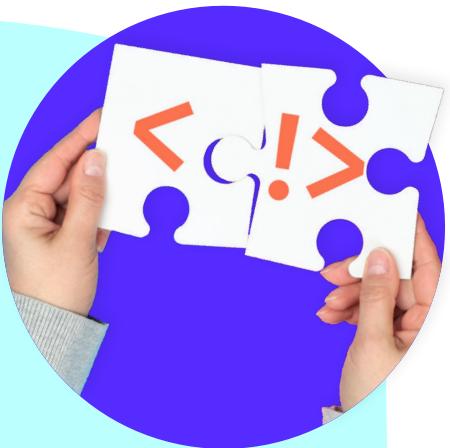
componentWillUpdate

render

componentDidUpdate

Unmounting (Desmontagem)

componentWillUnmount



Usando o comando **npx create-react-app**:

01. Crie um **projeto** em sua máquina.

<title>

MÃO + + NA MASSA

</title>

02. Em um documento separado, crie a estrutura de um botão (sem funcionalidade) e **importe-o para App.jsx** para que seja renderizado na tela.

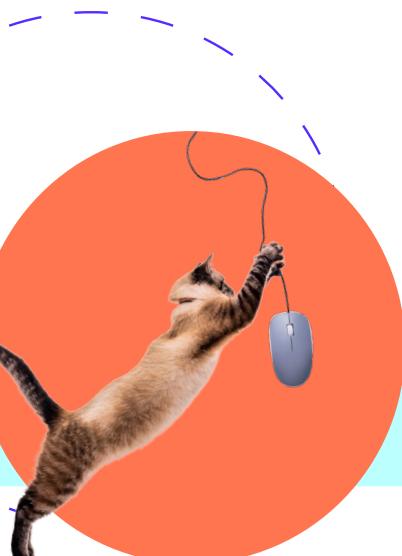
03. Use a documentação do React de referência.



{02.}

Criando o primeiro componente

O arquivo main.jsx é responsável pelo processo de renderização, enquanto o arquivo App.jsx será a espinha dorsal do projeto em que uniremos todos os componentes periféricos, isso para que desempenhem os seus papéis da maneira desejada.



<title>

PULO DO GATO

</title>

- Separe os componentes React em pastas descritivas que façam sentido para o projeto.



Alguns exemplos de tipos de componentes comuns, ou nomes de pastas, são os components, em que guardam os componentes que carregam a UI da aplicação, além das regras de negócio e partes lógicas.

É interessante sempre revisar os componentes em relação à sua complexidade. O ideal é termos um componente o mais desacoplado possível. Isso pode ser feito separando um componente em parte lógica e parte de visualização, por exemplo.

Outro exemplo são os adapters ou adaptadores (ou providers), que têm a funcionalidade de realizar as chamadas de APIs ou via websockets ou qualquer compartilhamento de dados com um serviço externo.

Há outro tipo a ser levado em conta que são os styles ou estilos, eles são responsáveis pela parte de CSS da aplicação. Além de CSS, pode ser utilizada a biblioteca de styled-components que é muito conhecida.

Dentro dos styles, é possível fazer a separação entre os estilos específicos para cada componente ou também criar os estilos globais para a aplicação.

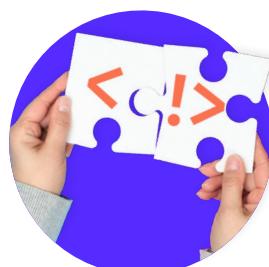
<title>

MÃO + +
NA MASSA

</title>

Comece seu app To Do, criando os seguintes elementos:

- › título
- › botão de adicionar tarefa
- › item da lista de tarefa





{03.}

Propriedades

Props é a abreviação para *properties* ou propriedades, são informações que podem ser passadas para um componente, podendo ser strings, numbers ou até funções, de forma que o componente poderá utilizá-las.

Para definir quais props desejamos passar para um componente, devemos inseri-las como atributos no local onde o componente está sendo utilizado.

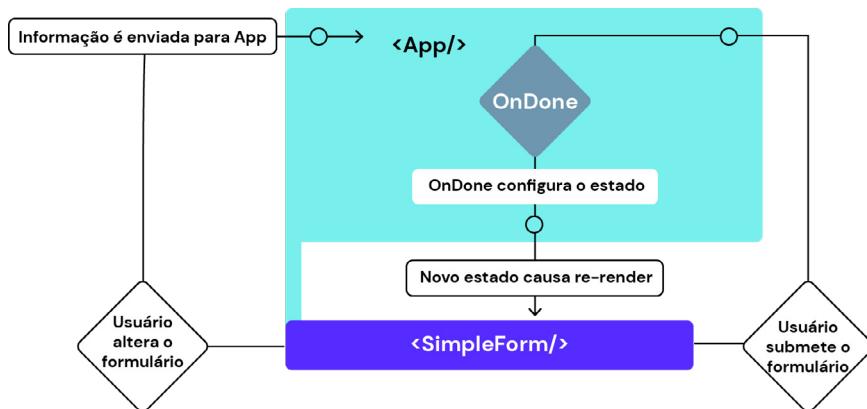
Os componentes em React possuem uma característica em relação às suas props, que é chamada de **Unidirectional data flow** (fluxo de dados unidirecional).





Esse comportamento possui algumas vantagens, como facilitar o debugging, pois quando o desenvolvedor sabe de onde o dado vem e para onde ele vai, pode-se usar ferramentas e testes para encontrar problemas de maneira mais eficiente.

Fluxograma respeitando o unidirectional data flow





<title>

PULO DO GATO

</title>

Caso queira utilizar a informação contida entre as tags de abertura e de fechamento do elemento, utilize **props.children**.

```
1  export function Button(props) {  
2      return (  
3          <button>{props.children}</button>  
4      )  
5  }
```

1 <Button>Teste</Button>

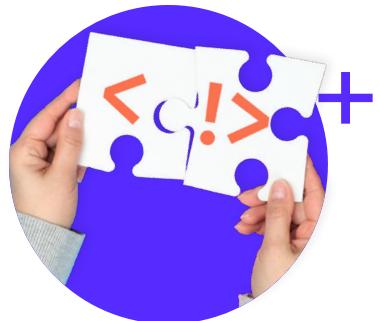
Teste

<title>

MÃO + + + NA MASSA

</title>

Insira props em seu componente Task.





{04.}

Criando o segundo componente

Antes de atribuirmos as funcionalidades para o App e para o Task, é preciso criar o último componente ainda de forma estática.

Ele vai ser chamado de `createTaskModal` e será uma espécie de pop-up que deverá abrir na tela quando escolhermos a opção “Adicionar uma nova tarefa +”. Ele será parecido com a imagem abaixo:



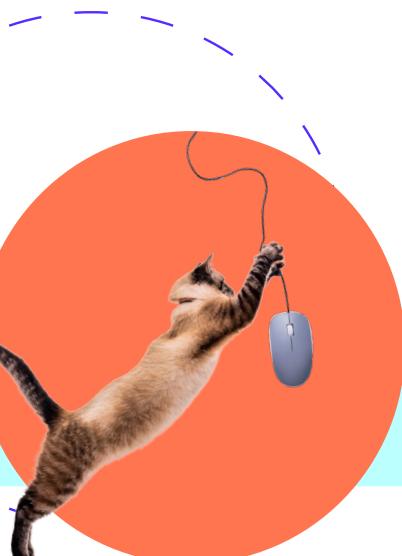


O componente Modal renderiza sua *children* (conteúdo interno) sobre um componente backdrop.

Oferecendo, assim, recursos importantes como:

- Cria um plano de fundo para desabilitar a interação abaixo do modal.
- Desativa a rolagem de conteúdo da página enquanto estiver aberto.
- Gerencia o foco, movendo-o para o modal e mantendo-o lá até que o modal seja fechado.
- Gerencia o empilhamento de camadas quando há mais de um modal.

O componente React Modal pode exercer inúmeras formas na página, como caixas de diálogo, drawers, menus ou popovers.



<title>

PULO DO GATO

</title>

- “Para facilitar sua reutilização, eu poderia separar meu componente em mais de um?”



<title>

MÃO + + NA MASSA

</title>



Crie e importe o seu CreateTaskModal.

“

Quanto mais difícil fica, mais próximo está o sucesso.

Chris Garrett

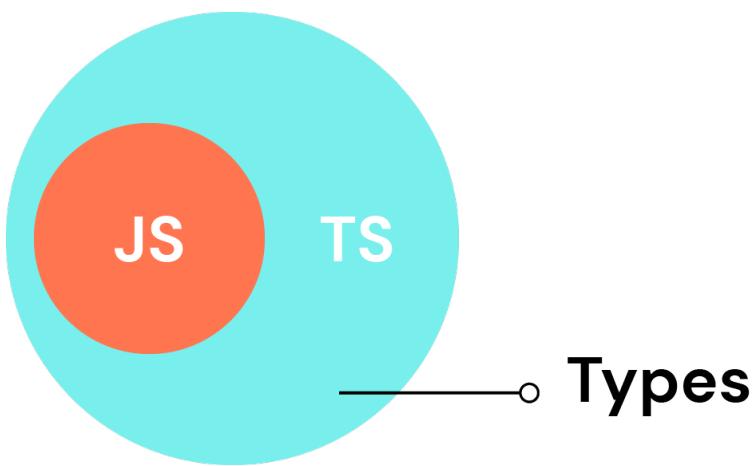
”



{05.}

Introdução ao TypeScript

Em 2012, a Microsoft desenvolveu uma nova linguagem que complementava o JavaScript, chamada Typescript.





Em JavaScript, os valores são tipados, pois um number é entendido como um number e uma string como uma string. Por outro lado, as variáveis não são tipadas.

Isso significa que uma variável que recebe uma string, na próxima linha de código pode guardar um boolean. Chamamos isso de tipagem dinâmica.

Já no Typescript, tanto os valores quanto as variáveis são tipadas.

Dessa forma, uma variável que guarda um boolean não pode guardar um number, por exemplo, sem gerar erro.

Esse tipo de linguagem tem uma tipagem estática ou forte, Inclusive, seu nome é “type” por ser a principal característica que ela agrupa ao JS.

A tipagem no TS pode ser declarada de forma explícita ou implícita.

No exemplo abaixo, é declarado de forma **explícita** que o tipo da variável número é do tipo number.

```
1 let numero: number;
2 numero = 10.5;
```



Devido à tipagem **estática**, se tentarmos atribuir um valor de outro tipo à variável número, teremos um erro.

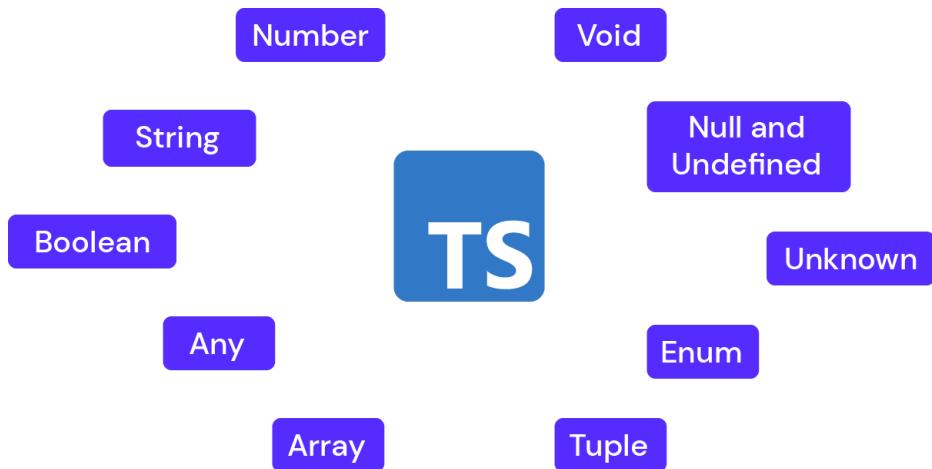
```
1 let numero: number;
2 numero = "frase"; //erro
```

A tipagem **implícita** ocorre pela inferência de tipo feita pelo Typescript. Ao atribuirmos uma string a uma variável e depois tentarmos inserir outro tipo de dado, teremos um erro. Isso, mesmo sem ter explicitamente declarado que ela deveria guardar um tipo string.

```
1 let fraseComum = "Frase que é uma string";
2 fraseComum = 42; //erro
```



Tipos mais utilizados em typescript



Um ponto importante é que o Typescript não diferencia inteiros de floats, ambos são tratados apenas como number, de forma que uma variável que teve um inteiro atribuído também pode ter um float atribuído em sequência.

Pensar em orientação a objetos com JavaScript pode deixar algumas dúvidas, fazendo com que não exploremos o potencial total que tem a orientação a objetos.

Porém, com TypeScript isso pode ficar muito mais claro e com mais possibilidades, como a utilização de classes, interfaces, encapsulamento e herança.



Classes

A declaração de classes em TypeScript é semelhante a de outras linguagens de programação. Permite criar seus atributos, gerar seu construtor, getters e setters e seus métodos.

Herança

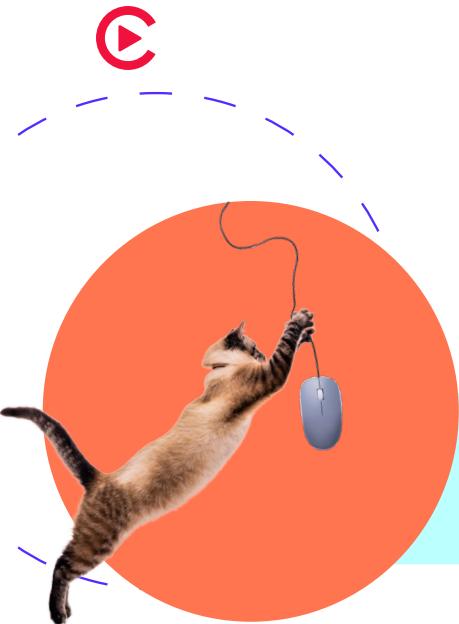
Assim como em JavaScript, utilizamos o comando **extends** para herdar uma classe de outra.

Encapsulamento

O TypeScript faz o uso de encapsulamento, semelhante ao JavaScript, utilizando getters e setters para ter acessos a atributos internos e/ou protegidos de uma classe.

Interfaces

As interfaces podem ser entendidas como um contrato que as classes filhas devem assinar com elas, de forma que as classes que as implementam devem realizar todos os métodos descritos pela interface.



<title>

PULO DO GATO

</title>

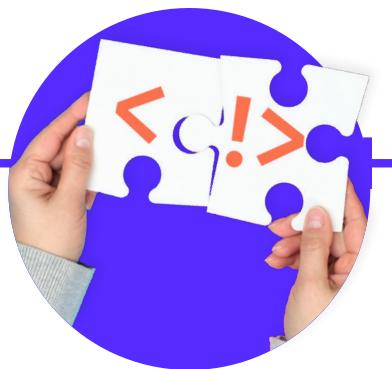
- A sintaxe do JSX mantém a mesma para o template em JavaScript ou TypeScript.

<title>

MÃO + + NA MASSA

</title>

Instale o TypeScript na sua máquina e faça a conversão do seu projeto de JS para TS.





<title>

DESAFIO CONQUER

</title>



```
if (challenge)  
  // start
```

<body>

- Crie um novo projeto usando npm `create vite@latest -template` em react-ts.
- Aproveitando a estrutura inicial da página principal em `App.tsx`, realoque o botão de `count` e toda a sua estrutura para um componente separado em uma nova pasta `components` e nomeie como `Counter`.
- Realoque outras partes, além do botão que fazem referência ao uso de states pelo Hook `useState`.
- Além de realocar o botão, incremente também um botão para decrescer o valor.
- Em vez de mostrar a contagem no próprio botão, mostre em um elemento `<h4>` que deve ficar entre os dois botões (o que incrementa e o que decrementa).
- Toda essa estrutura deve estar separada do arquivo `App.tsx`, e estar localizada no `index.tsx` dentro de `Counter`.

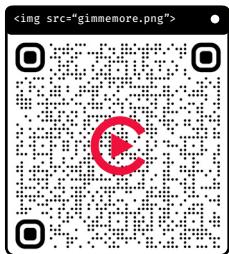
</body>



<title>

QUERO MAIS

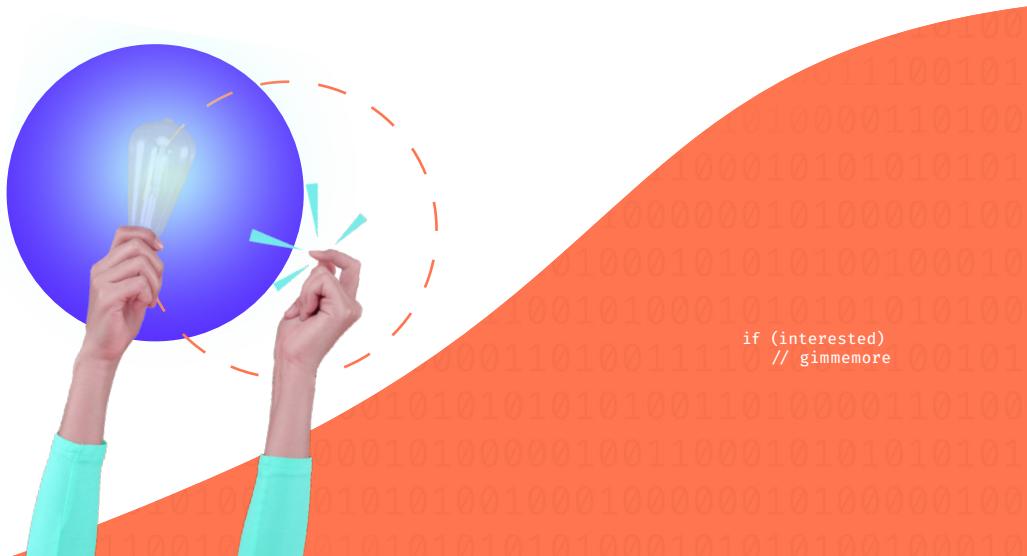
</title>



React re-renders



React modal





Anotações

-
-
-
-
-
-

