



React para iniciantes





<title>

AGENDA DO </> </> </> MÓDULO

</title>



Agenda do módulo



Anotações



{Estado e imutabilidade}

{Funcionalizar componentes da aplicação}

{Finalização de componentes}

{Finalização da Aplicação (To Do)}



<title>

MISSÃO DO MÓDULO

</title>

Aplicar **Hooks** a **functional components** em React e **finalizar** a aplicação To Do.

{01.}

Estado e imutabilidade

Alguns dos conceitos mais importantes de todo o espaçofuncionamento do React são:

- Imutabilidade.
- Hooks.
- States.





Alteração sem mutação = Imutabilidade

Props vs. states

As **props** (ou properties) são utilizadas em React para transmitir dados entre componentes.

Esses dados podem ser de qualquer tipo:

- string;
- number;
- array.

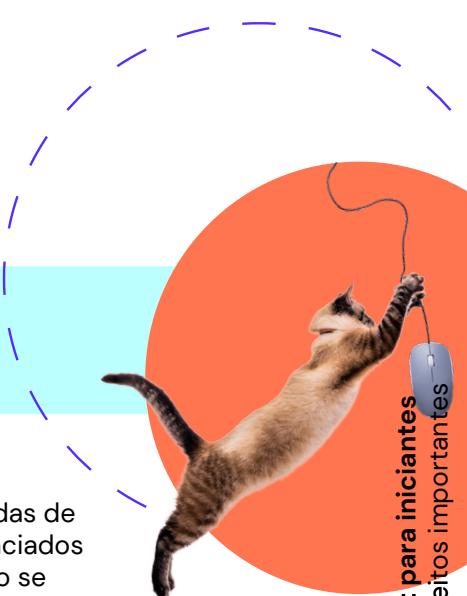
```
<title>
```

PULO DO GATO

```
</title>
```

- Sempre esteja atento ao criar um componente para que ele não altere suas **props**.

Diferentemente das props que são transmitidas de um componente a outro, os **states** são gerenciados dentro do próprio componente apenas, como se fossem variáveis declaradas dentro de uma função.





**Seja flexível,
mas mantenha suas
variáveis imutáveis!**



React para iniciantes
Conceitos importantes



A partir do **React 16.8**, surgiu um recurso para que os **functional components** possam manipular states, ao invés de apenas receber props e renderizá-las na tela.

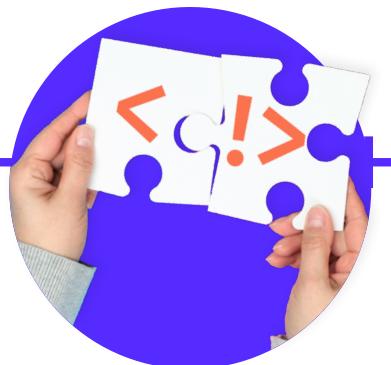
Essa funcionalidade é a dos **React Hooks**.

```
<title>
```

MÃO + NA MASSA

```
</title>
```

- Utilize os **Hooks** trabalhando com **states** e usando functional components.
- Seguindo os passos descritos na aula, faça o mesmo no seu arquivo pessoal do projeto.





{02.}

Funcionalizar componentes da aplicação

LocalStorage:

Mantém os dados ao fechar
a página.

SessionStorage:

Apaga os dados ao fechar
a página.



<title>

PULO DO GATO

</title>

- Verifique o que está guardado no localStorage em:
Application > Storage > localStorage.

Minhas tarefas

Adicionar uma nova tarefa +

localStorage

Key	Value
tarefas	[{"id": 1, "title": "Tarefa 1", "isCompleted": false}, {"id": 2, "title": "Tarefa 2", "isCompleted": false}]



Em **types.ts** é possível criar interfaces, de forma que, ao serem implementadas, tornam necessário seguir o contrato de tipos definidos por elas.

<title>

MÃO NA MASSA

</title>

- Crie sua interface para as tasks e defina como buscá-las do **localStorage**.
- Seguindo os passos descritos na aula, faça o mesmo no seu arquivo pessoal do projeto.



{03.}

Finalização de componentes

A **handleToggleTask** é responsável por receber, como prop, a task que foi clicada. Então, a função irá procurar essa task no **localStorage**.

Sempre que estiver trabalhando com funções ou props de um componente separado, haverá quatro momentos em que você deve utilizá-lo ou chamá-lo.

Dê uma olhada no pulo do gato:



<title>

PULO DO GATO

</title>

- > Declarando (criando) a função ou prop e especificando o que ela irá realizar.
- > Criando sua interface no **types.ts**.
- > Implementando-a na **index.tsx** do seu componente.
- > Implementando-a no momento em que o componente é invocado no **App.tsx** ou na página que está importando-o.

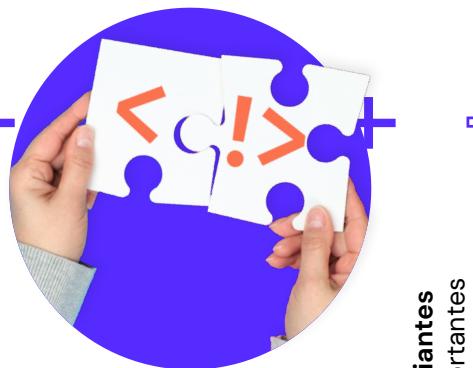


<title>

MÃO + + + NA MASSA

</title>

- Crie as funções **handleToggleTask** e **handleRemoveTask**, criando suas interfaces de types e implementando-as na aplicação.
- Seguindo os passos descritos na aula, faça o mesmo no seu arquivo pessoal do projeto.





{04.}

Finalização da Aplicação (To Do)

SetStateAction

SetStateAction funciona como uma sobrecarga vista em outras linguagens de programação. Ao usar essa função, é possível setar seu state diretamente, ou usar a função com o state anterior como argumento.

Dispatch

Por sua vez, o **Dispatch** indica que a função que está dentro dele não está retornando nenhum dado ou seja, void.

Void significa “espaço totalmente vazio”; quando usado em programação, refere-se a um retorno de “nada” – um “valor vazio”.



<title>

PULO DO GATO

- Persistir é quando o estado de uma variável sobrevive ao processo que a criou.

</title>

UseEffect

Para que as tarefas não se apaguem a cada atualização, vamos contar com um outro Hook que é o **useEffect**.

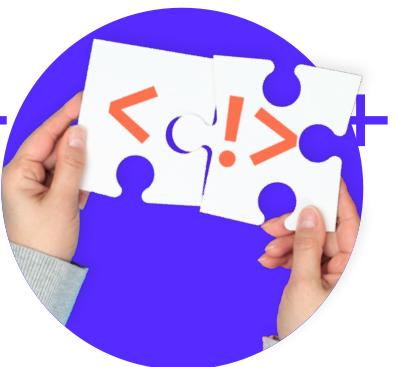
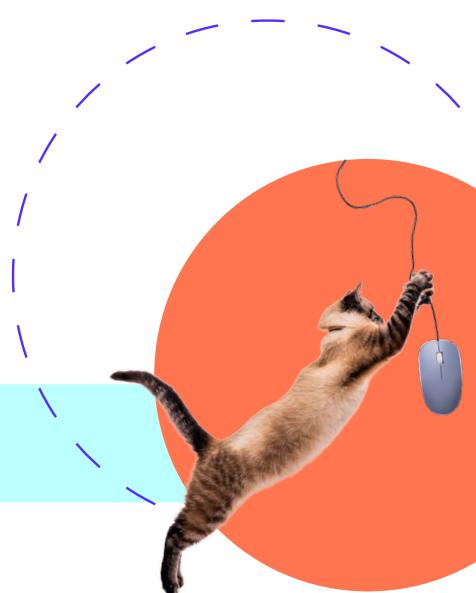
O **useEffect** diz ao React que o componente precisa executar algo logo após a renderização, tanto a primeira quanto qualquer outra renderização.

<title>

MÃO + + + NA MASSA

</title>

- Finalize seu componente **CreateTaskModal** implementando **onRequestClose**, criando **handleCreateNewTask** e implementando-a, além de criar a maneira de fazer persistir as tasks no localStorage.
- Seguindo os passos descritos na aula, faça o mesmo no seu arquivo pessoal do projeto.





<title>

DESAFIO CONQUER

</title>



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

<body>

Usando os Hooks **useState** e **useEffect**:

- Crie um projeto do zero, usando `npm create vite@latest – template` em React e TypeScript.
- Aproveite o layout fornecido, mas troque o botão de counter da página inicial por um componente Counter que você deve criar.
- Este componente Counter deve ser um counter para incrementar e decrementar o valor, mas insira uma trava de maneira que, quando o valor for igual a zero, o botão de decremento não funcione.
- Usando **useEffect**, mostre esse valor multiplicado por uma constante de sua escolha logo abaixo do Counter.

</body>



<title>

DESAFIO CONQUER

</title>



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

<body>

Vite + React

- 2 + 10

Edit `src/App.tsx` and save to test HMR

Click on the Vite and React logos to learn more

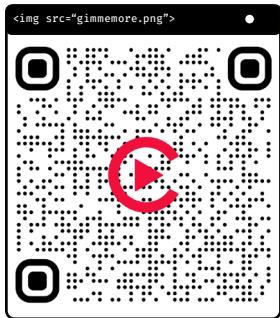
</body>



<title>

QUERO MAIS

</title>



Assista ao vídeo:

Apresentação Hooks
para Comunidade Dev



```
if (interested)  
// gimmemore
```



Anotações

-
-
-
-
-
-

