# 1. Atividades Práticas - React Hooks

---

Nesta atividade prática, vamos trabalhar com os conceitos de Rect Hooks.  Nesta atividade prática, vamos desenvolver uma aplicação de Todo-List.

## 1.1 Criando nosso Projeto

Vamos criar um projeto chamado my-todo, digitando o seguinte comando no terminal:

```
yarn create react-app my-todo
```

O processo leva alguns segundos e, logo após terminar, digite o seguinte comando para entrar no diretório recém criado do nosso projeto:

```
cd my-todo
```

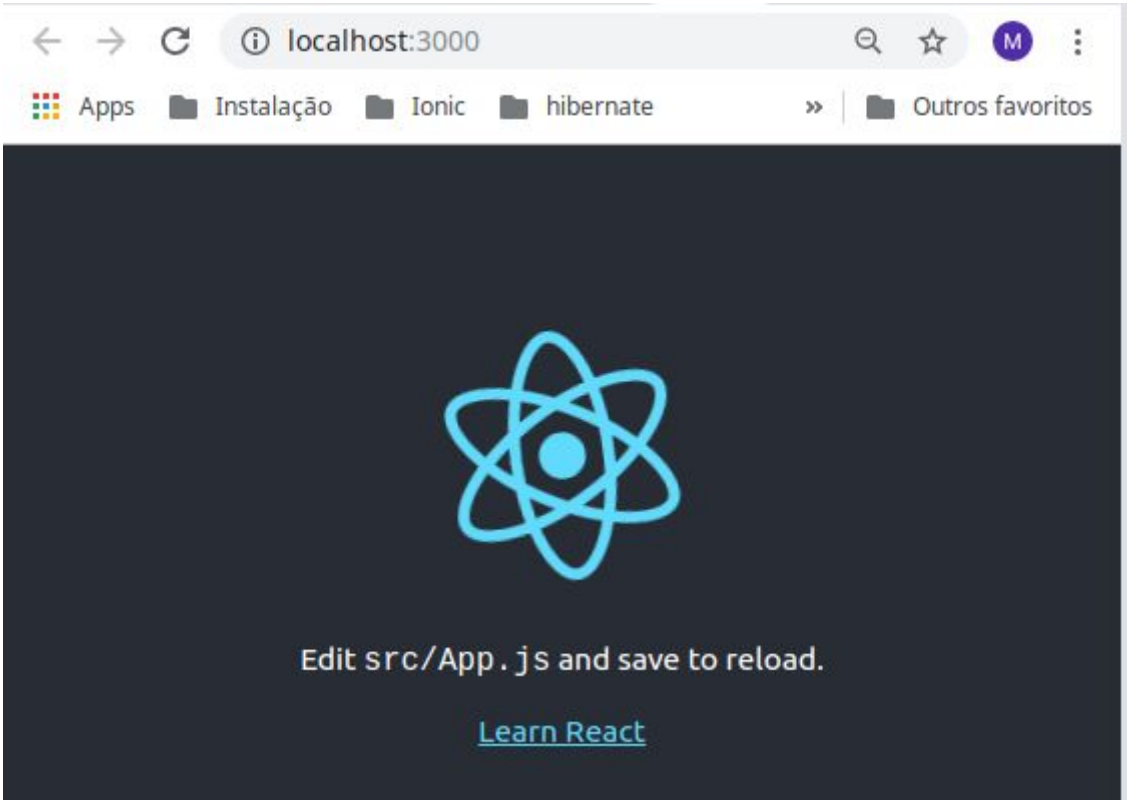
Você pode abrir a aplicação no editor Visual Studio Code. Para isso, dentro do diretório my-todo digite o seguinte comando:

```
code .
```

O Visual Studio Code deverá abrir com a pasta my-todo sendo acessada. Agora, vamos testar nossa aplicação. Digite a seguinte linha de comando:

```
yarn start
```

Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:3000




A página será recarregada automaticamente se você fizer alterações no código.

## 1.2 Criando um estilo para sua aplicação

Agora, vá para o arquivo src/App.css e adicione os seguintes estilos CSS que usaremos em todo o aplicativo. Cabe lembrar que o CSS não será o foco desta aula. Veja como deve ficar:

```
.app {
 background: #209cee;
 padding: 30px;
 height: 100vh;
}

.todo-list {
 background: #e8e8e8;
 border-radius: 4px;
 padding: 5px;
 max-width: 400px;
}

.todo {
```

```
 background: #fff;
 box-shadow: 1px 1px 1px rgba(0, 0, 0, 0.15);
 padding: 3px 10px;
 font-size: 12px;
 margin-bottom: 6px;
 border-radius: 3px;
 display: flex;
 align-items: center;
 justify-content: space-between;
}
button {
 background: rgb(16, 241, 211);
 box-shadow: 1px 1px 1px rgba(0, 0, 0, 0.15);
 padding: 3px 10px;
 font-size: 12px;
 margin-bottom: 6px;
 border-radius: 3px;
 align-items: center;
 justify-content: space-between;
 cursor: pointer;
}
```

## 1.3 Lendo uma lista de itens

Com seu aplicativo em execução e o estilo pronto para ser usado, vamos começar na parte ler uma lista de itens do Todo-List. Ou seja, queremos fazer uma lista de coisas para que possamos ler/visualizar a lista.

### 1.3.1 Adicionando no estado interno

Entre no seu arquivo src/App.js e adicione um estado ao nosso componente. Como usaremos React Hooks, o estado parecerá um pouco diferente do que usamos com classes.

```
import React, {useState} from 'react';
import './App.css';

function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React" },
   { text: "Encontrar um amigo para o almoço" },
   { text: "Passar no supermercado" }
```

```
 ]);
 return (
   <div>
   </div>
 );
}
export default App;
```

Observe que componente que criamos é um componente funcional. Nas versões anteriores do React, os componentes funcionais não conseguiam lidar com o estado. Mas agora, usando Hooks, eles podem.

Desta forma, em nosso exemplo:

- O primeiro parâmetro, `todos`, usamos para nomear nosso estado.
- O segundo parâmetro, `setTodos`,usamos para definir o estado.

O hook `useState` é o que o React usa para se conectar ao estado do componente. Em seguida, criamos um array de objetos e temos o início de nosso estado. Ainda, observe que estamos com a `<div>` vazia (neste momento).

### 1.3.2 Comparando com um componente de classe

Antes de continuar, observe como ficaria a criação deste componente com classes:

```
class App extends Component {
   state = {
     todos: [
       { text: "Aprender sobre o React" },
       { text: "Encontrar um amigo para o almoço" },
       { text: "Passar no supermercado" }
     ]
   }
    setTodos = todos => this.setState({ todos });
    render() {
     return <div></div>
   }
  }
```

O React Hooks permite um código muito mais limpo, certo?

Mas, vamos voltar para a nossa aplicação.

### 1.3.3 Criando o componente Todo

Após, queremos criar um componente que possamos usar posteriormente no retorno do componente App. Cabe lembrar que podemos criar diversos componentes em um mesmo arquivo. Desta forma, vamos criar o componente Todo no arquivo App.js. Este componente mostrará a parte "`text`" do valor de `todo` recebido (`todo.text`), assim:

```
import React, {useState} from 'react';
import './App.css';

const Todo = ({ todo }) => <div className="todo">{todo.text}</div>;

function App() {
 ...
}
```

### 1.3.4 Obtendo a lista de itens de Todo

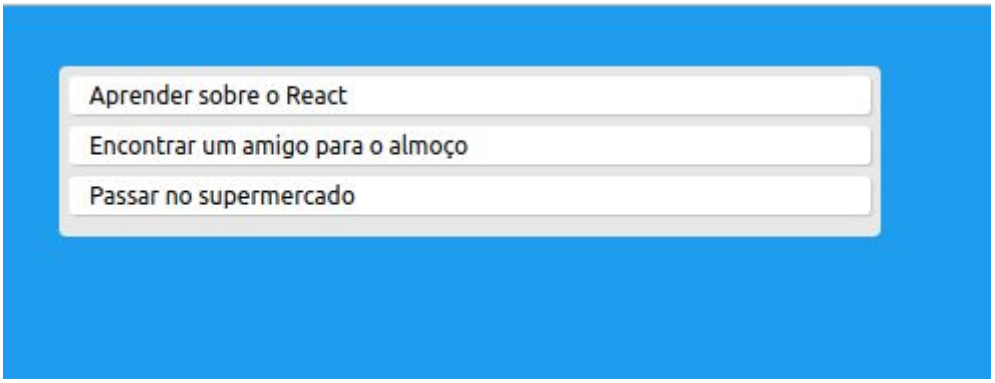
Em seguida, devemos instanciar o componente Todo no componente `App`. Logo, desça até a parte de retorno do componente `App`, onde temos `<div></div>`. Queremos que a nossa lista seja exibida na página.

```
import React, {useState} from 'react';
import './App.css';

const Todo = ({ info }) => <div className="todo">{info.text}</div>;

function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React" },
   { text: "Encontrar um amigo para o almoço" },
   { text: "Passar no supermercado" }
 ]);
 return (
   <div className="app">
     <div className="todo-list">
       {todos.map((todo, index) => (
         <Todo key={index} index={index} todo={todo} />
       ))}
     </div>
```

```
 </div>
 );
}
```

Assim, usando o método JavaScript chamado `map()`, conseguimos criar um novo array de itens, mapeando os itens do estado interno na variável `todo` e exibindo-os de acordo com seu índice.

Veja no navegador o resultado até o momento:




### 1.3.5 Criando novos itens na lista de tarefas

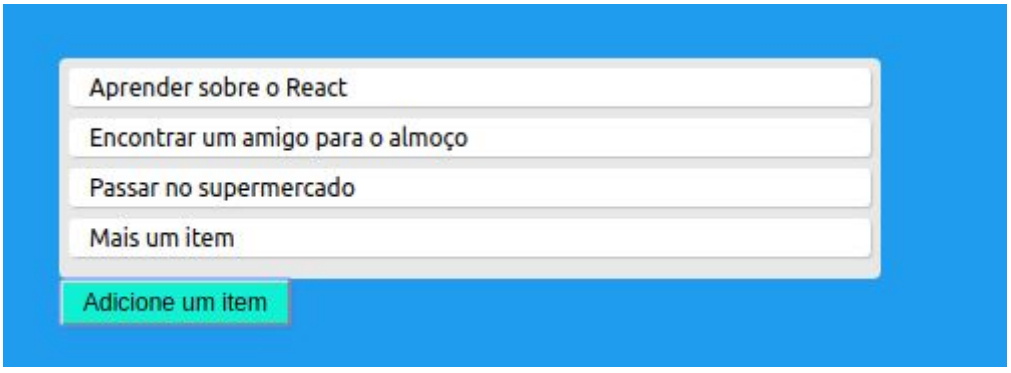
Vamos codificar a parte da aplicação que permite criar um novo item na lista de tarefas. Assim, vamos criar a função `addTodo` dentro do componente `App`. Permanecendo no App.js, essa a função deverá pegar a lista de itens existente, adicionar o novo item e exibir essa nova lista. Vamos criar um botão para adicionar um item qualquer, neste caso “Mais um item”.

```
import React, {useState} from 'react';
import './App.css';

const Todo = ({ todo }) => <div className="todo">{todo.text}</div>;

function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React" },
   { text: "Encontrar um amigo para o almoço" },
   { text: "Passar no supermercado" }
 ]);
 const addTodo = info => {
   const newTodos = [...todos, { text: info }];
   setTodos(newTodos);
 };
```

```
 return (
   <div className="app">
     <div className="todo-list">
       {todos.map((todo, index) => (
         <Todo key={index} index={index} todo={todo} />
       ))}
     </div>
     <div>
       <button onClick={() => addTodo("Mais um item")}>
           Adicione um item
       </button>
     </div>
 </div>
 );
}

export default App;
```

Observe que usamos o spread operator. Os três pontos antes de `todos` copia os itens da lista, para que possamos adicionar o novo item de tarefa. Em seguida, atualizaremos o estado com `setTodos`.

Agora você pode adicionar um item de tarefa à sua lista no seu navegador.




### 1.3.6 Atualizando itens na lista de tarefas

Vamos, agora, adicionar a funcionalidade para “riscar” um item em nossa lista de tarefas. Para realizar isso, o estado em nosso componente `App` precisa de uma informação sobre o status de cada item, uma vez que precisamos informar que o status esteja "Concluído". Logo, ao adicionar a informação `isCompleted` (que inicia for igual a `false`) ao array.

```
import React, {useState} from 'react';
import './App.css';
```

```
const Todo = ({ todo }) => <div className="todo">{todo.text}</div>;

function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
   const newTodos = [...todos, { text: info, isCompleted: false }];
   setTodos(newTodos);
 };
 ...
```

Observe que também precisamos alterar a função `addTodo` para adicionar esta informação sobre o `status`.

Agora, dentro do componente App, vamos precisar de uma nova função, chamada de `completeTodo`, que seja capaz de "concluir" um item. Vamos adicionar esta função no componente funcional `Todo`. Para isto, usar o operador spread para pegar a lista atual de itens. Nesta função, alteraremos o status `isCompleted` para `true`, para que ele saiba que um item foi concluído. Ele atualizará o estado e o definirá para o `newTodos`.

```
 ...
 const completeTodo = index => {
       const newTodos = [...todos];
       newTodos[index].isCompleted = true;
       setTodos(newTodos);
   };
 ...
```

Veja que, quando o botão Concluir é clicado, ele adiciona o estilo CSS de decoração de texto e riscará o item. Estamos usando um operador ternário, um recurso no JavaScript ES6, que é outra maneira de fazer uma declaração `if/else`. Assim, podemos “concluir” um item na lista e "atualizar" esta lista.

```
import React, {useState} from 'react';
import './App.css';
```

```
function Todo({ todo, index, completeTodo }) {
 return (
  <div className="todo"
    style={{ textDecoration: todo.isCompleted ? "line-through" : "" }}>
    {todo.text}
    <div>
       <button onClick={() => completeTodo(index)}>Complete</button>
    </div>
  </div>
 );
}

function App() {
...
}
export default App;
```

Em seguida, desça até o retorno do componente `App` e adicione a informação que passe `completeTodo` para o componente `Todo`:

```
function App() {
  ...
 return (
   <div className="app">
     <div className="todo-list">
       {todos.map((todo, index) => (
         <Todo key={index}
               index={index}
               todo={todo}
               completeTodo={completeTodo}
         />
       ))}
     </div>
   ...
```

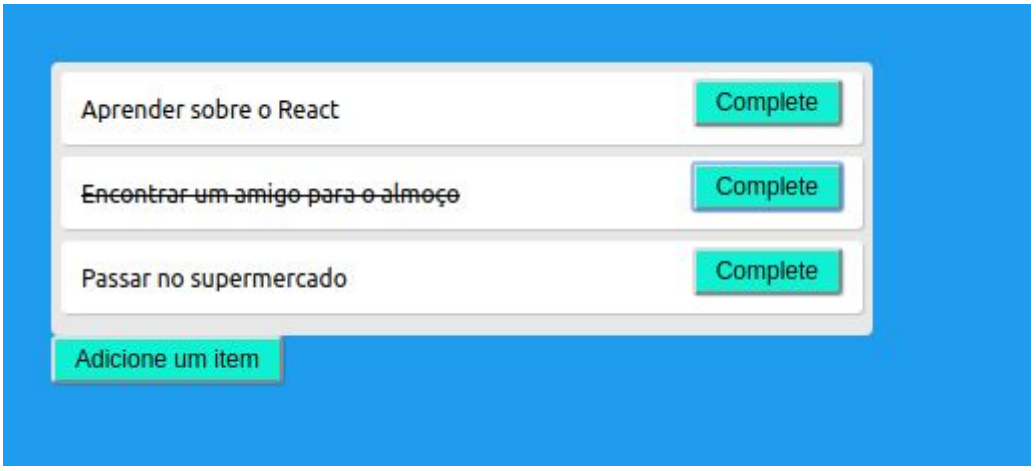
Ou seja, até o momento, nosso código deve estar assim:

```
import React, {useState} from 'react';
import './App.css';

function Todo({ todo, index, completeTodo }) {
```

```
 return (
   <div className="todo"
       style={{ textDecoration: todo.isCompleted ? "line-through" : ""
}}>
     {todo.text}
     <div>
       <button onClick={() => completeTodo(index)}>Complete</button>
     </div>
   </div>
 );
}

function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
   const newTodos = [...todos, { text: info, isCompleted: false }];
   setTodos(newTodos);
 };
 const completeTodo = index => {
   const newTodos = [...todos];
   newTodos[index].isCompleted = true;
   setTodos(newTodos);
 };

 return (
   <div className="app">
     <div className="todo-list">
       {todos.map((todo, index) => (
         <Todo key={index}
               index={index}
               todo={todo}
               completeTodo={completeTodo}
         />
```

```
        ))}
      </div>
      <div>
        <button onClick={() => addTodo("Mais um item")}>
            Adicione um item
        </button>
      </div>
  </div>
  );
}

export default App;
```

Veja no navegador que sua lista de tarefas deve estar funcionando!




Agora podemos ler nossa lista, adicionar um item à nossa lista e atualizar o status completo de cada item. Em seguida, adicionaremos a funcionalidade de exclusão.

## 1.4 Excluindo um item da lista de tarefas

Dentro do componente `App`, vamos criar a função `removeTodo` para que, quando clicarmos em um "X" para excluir um item, ele seja excluído. Nesta função `removeTodo`, usaremos novamente o operador spread. Mas, assim que pegarmos a lista atual, separaremos o índice escolhido do array de itens. Uma vez removido, retornaremos o novo estado, configurando-o para ser `newTodos`, através de `setTodos`.

```
...
const removeTodo = index => {
   const newTodos = [...todos];
   newTodos.splice(index, 1);
```

```
    setTodos(newTodos);
  };
...
```

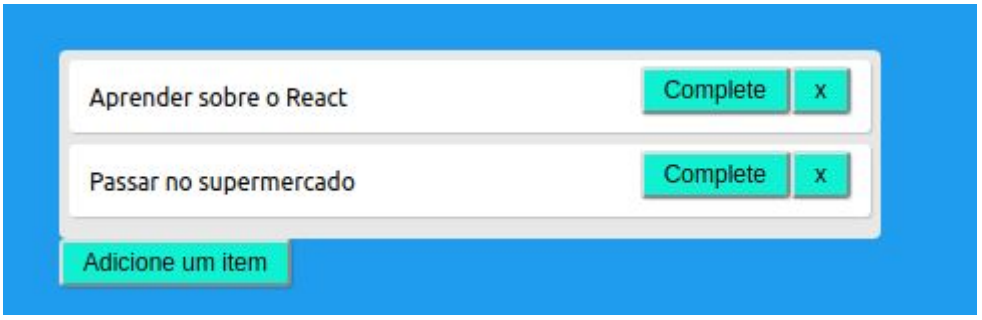
Em nosso componente Todo, você precisamos adicionar um botão que chame `removeTodo`.

```
...
function Todo({ todo, index, completeTodo, removeTodo }) {
  return (
    <div className="todo"
        style={{ textDecoration: todo.isCompleted ? "line-through" : ""
}}>
      {todo.text}
      <div>
        <button onClick={() => completeTodo(index)}>Complete</button>
        <button onClick={() => removeTodo(index)}>x</button>
      </div>
    </div>
  );
}
...
```

Em seguida, desça até o retorno do componente `App` e adicione a informação que passe `removeTodo` para o componente `Todo`:

```
...
  return (
    <div className="app">
      <div className="todo-list">
        {todos.map((todo, index) => (
          <Todo key={index}
                index={index}
                todo={todo}
                completeTodo={completeTodo}
                removeTodo={removeTodo}
          />
        ))}
      </div>
...
```

Com isso adicionado, acesse seu navegador e você verá um botão com um "X" que, quando clicado, exclui completamente o item.




O nosso código final é o seguinte:

```
import React, {useState} from 'react';
import './App.css';

function Todo({ todo, index, completeTodo, removeTodo }) {
 return (
   <div className="todo"
    style={{ textDecoration: todo.isCompleted ? "line-through" : "" }}>
     {todo.text}
     <div>
       <button onClick={() => completeTodo(index)}>Complete</button>
       <button onClick={() => removeTodo(index)}>x</button>
     </div>
   </div>
 );
}
function App() {
 const [todos, setTodos] = useState([
   { text: "Aprender sobre o React",
     isCompleted: false },
   { text: "Encontrar um amigo para o almoço",
     isCompleted: false },
   { text: "Passar no supermercado",
     isCompleted: false }
 ]);
 const addTodo = info => {
   const newTodos = [...todos, { text: info, isCompleted: false }];
   setTodos(newTodos);
 };
```

```
 const completeTodo = index => {
   const newTodos = [...todos];
   newTodos[index].isCompleted = true;
   setTodos(newTodos);
 };
 const removeTodo = index => {
   const newTodos = [...todos];
   newTodos.splice(index, 1);
   setTodos(newTodos);
 };
 return (
   <div className="app">
     <div className="todo-list">
       {todos.map((todo, index) => (
         <Todo key={index}
               index={index}
               todo={todo}
               completeTodo={completeTodo}
               removeTodo={removeTodo}
         />
       ))}
     </div>
     <div>
       <button onClick={() => addTodo("Mais um item")}>
           Adicione um item
       </button>
     </div>
 </div>
 );
}
export default App;
```

Desta forma, nesta atividade prática você criou uma aplicação de lista de tarefas com React Hooks