




escola
britânica de
artes criativas
& tecnologia

Desenvolvedor Full Stack Python

Introdução ao React

Introdução

Durante este módulo conhecemos os fundamentos do React, como criar aplicações e componentes com uso de estado e propriedades, além disso conhecemos o CSS Modules, uma nova forma de criar código de estilos pensando em acoplamento e estilos restritos ao contexto de um componente.

 [Clique aqui](#) para consultar o código do projeto desenvolvido durante o módulo.

Introdução

O **ReactJS** é uma biblioteca para construção de interfaces **Web**, ele foi criado pelo Facebook, agora Meta, em 2013.

O React é hoje o principal framework no mundo frontend, empresas como Microsoft, Netflix, Amazon, AirBnb fazem uso desta tecnologia.

Além de aplicativos para Web, podemos criar aplicativos para Android e iOS com uma variante do React, o React Native.

Introdução ao JSX

O React trabalha com uma sintaxe própria chamada **JSX**. Esta sintaxe junta o JavaScript com XML, que é basicamente uma estrutura de tags, parecida com o HTML.

Os componentes React são funções JavaScript que retornam uma estrutura de tags, também chamada de **Elemento React**.

Introdução ao JSX - exemplo

```
function Titulo() {  
  const nome = "Lucas";  
  return (  
    <h1>Bem-vindo, {nome}</h1>  
  )  
}  
export default Titulo
```

Podemos escrever código JavaScript dentro da estrutura de tags, com o uso das chaves `{ }`

O código acima, é um exemplo de componente e pode se utilizado por qualquer outro arquivo .jsx que o importe.

Introdução ao JSX - exemplo

Caso precisemos exportar mais de uma tag, é necessário incluir num wrapper uma tag que irá encapsular todo o conteúdo:

```
function Titulo() {  
  const nome = "Lucas";  
  return (  
    <div>  
      <h1>Bem-vindo, {nome}</h1>  
    </div>  
  )  
}  
export default Titulo
```

Introdução ao JSX - exemplo

Para não inserirmos a `div` em tudo, o React disponibiliza **fragments**, que são tags vazias, apenas para encapsulamento:

```
function Titulo() {  
  const nome = "Lucas";  
  return (  
    <>  
      <h1>Bem-vindo, {nome}</h1>  
    </>  
  )  
}  
export default Titulo
```

Componentes

Como visto anteriormente, um componente é basicamente uma função JavaScript que retorna um elemento React.

Para utilizar um componente, importamos ele no arquivo que irá fazer o uso e depois disso escrevemos ele como um tag:

```
import Titulo from './components/Titulo.jsx';  
function App() {  
  return (  
    <Titulo />  
  )  
}
```


Componentes

Por convenção os nomes das funções e dos componentes começa com a inicial maiúscula, esta é a recomendação.

Os componentes podem receber propriedades, também chamada de **props**, que são basicamente atributos incluídos na tag do componente.

```
<Titulo nome="Maria" />
```

Para acessar uma propriedade:

```
function Titulo(props) {  
  return (  
    <h1>Olá, {props.nome}</h1>  
  )  
}
```

Estado

O estado de um componente é basicamente um valor que poderá ser alterado e esta alteração irá implicar em mudanças na interface, na UI.

Pode ser um número de um contador que será incrementado, um nome que ao usuário preencher o campo será exibido instantaneamente.

Para criar um estado no React utilizamos uma função chamada **useState**.

Estado

O **useState** é uma função que nos retornará um valor (estado) e uma função para alterar este valor:

```
const [nome, alteraNome] = useState('');
```

Dentro da função useState passamos o valor inicial, no caso uma string vazia.

Não podemos, nem devemos alterar o valor diretamente, como nome = "João", isso fere os princípios de reatividade do React e caso utilizemos este valor dentro das tags em return, a UI pode não ser atualizada.

Costumamos nomear a função que altera o valor com set antes, por exemplo: setNome

Eventos

O **React** possui todos os eventos que o JavaScript possui, porém em camelCase:

```
<input type="text" onChange={ } />
```

Dentro do evento podemos executar a função diretamente ou passar uma função.

```
<input  
  type="text"  
  onChange={evento => setNome(evento.target.value)}  
/>
```

Ou com a função:

```
function alterarNome(evento) {  
  setNome(evento.target.value);  
}
```

```
<input type="text" onChange={alterarNome} />
```

useEffect

O **useEffect** nos possibilita executar uma ação após determinado evento, por exemplo executar um código ao montar o componente, executar uma ação ao alterar um estado.

Quando utilizamos o **useEffect** sem argumentos ele será executado a cada mudança no estado do componente:

```
useEffect(() => {  
  console.log("o componente foi atualizado");  
})
```

useEffect

Para fazer com que o **useEffect** seja executado apenas quando determinado dado for alterado, devemos passar um array como segundo argumento e dentro deste array informar o dado que iremos observar:

```
useEffect(() => {  
  console.log("o nome foi atualizado");  
}, [nome])
```

```
useEffect(() => {  
  console.log("as props foram atualizadas");  
}, [props])
```

useEffect

Quando passamos um array vazio, o **useEffect** será executado ao montar o componente.

```
useEffect(() => {  
  console.log("o componente foi atualizado");  
}, [])
```

Tanto o **useState** como **useEffect** devem estar dentro da função do componente, mas fora do return.

useEffect

```
function Formulario() {  
  const [idade, setIdade] = useState(0);  
  
  useEffect(() => {  
    console.log("a idade mudou");  
  }, [idade]);  
  
  return (  
    <input  
      type="number"  
      onChange={e => setIdade(e.target.value)}  
    />  
  )  
}
```


Trabalhando com listas

Para renderizar dados brutos como números, strings e propriedades de objetos no JSX, utilizamos o **conjunto de chaves**, porém isso não funciona quando temos que renderizar itens de um array.

Para renderizar vários itens precisamos utilizar o **map** e retornar para cada item do array um elemento React.

Trabalhando com listas

```
function Produtos() {  
  const items = ["Celular", "TV", "PS5"]  
  
  return (  
    <>  
      <h3>Produtos</h3>  
      <ul>  
        {items.map(item => (  
          <li key={item}>{item}</li>  
        ))}  
      </ul>  
    </>  
  )  
}
```

Trabalhando com listas

No exemplo anterior temos o atributo `key` na `LI`, isso é necessário para que o React saiba gerenciar os elementos renderizados a partir de uma iteração.

Geralmente utilizamos um `ID` que irá vir de uma integração com o back-end. Caso não possuímos um `ID` devemos utilizar algum valor único, que não se repita na lista.

CSS Modules

No React podemos utilizar o CSS comum, importamos ele no código JavaScript:

```
import './main.css';
```

Porém podemos ter casos de colisões onde é necessário o uso da palavra **importante**, com o CSS Modules criamos um CSS para cada componente.

CSS Modules

Um arquivo CSS Modules deve determinar com **.module.css** e fazemos a importação dele como se fosse um código JavaScript:

```
import estilos from './botao.module.css';
```

E ao utilizar a classe CSS passamos ela no atributo `className`:

```
<button className={estilos.botaoPrincipa}>
```

CSS Modules

Passamos o nome da classe como se fosse uma propriedade de objeto, não podendo utilizar o traço para separar, utilizamos o padrão camelCase.

Além disso no React não podemos utilizar simplesmente `class=""` a palavra `class` já existe no JavaScript e está relacionada a orientação a objetos, por isso temos o `className` para evitar conflitos.