

# DBStart

React

Instrutor: Júlio Pereira Machado ([julio.machado@pucrs.br](mailto:julio.machado@pucrs.br))

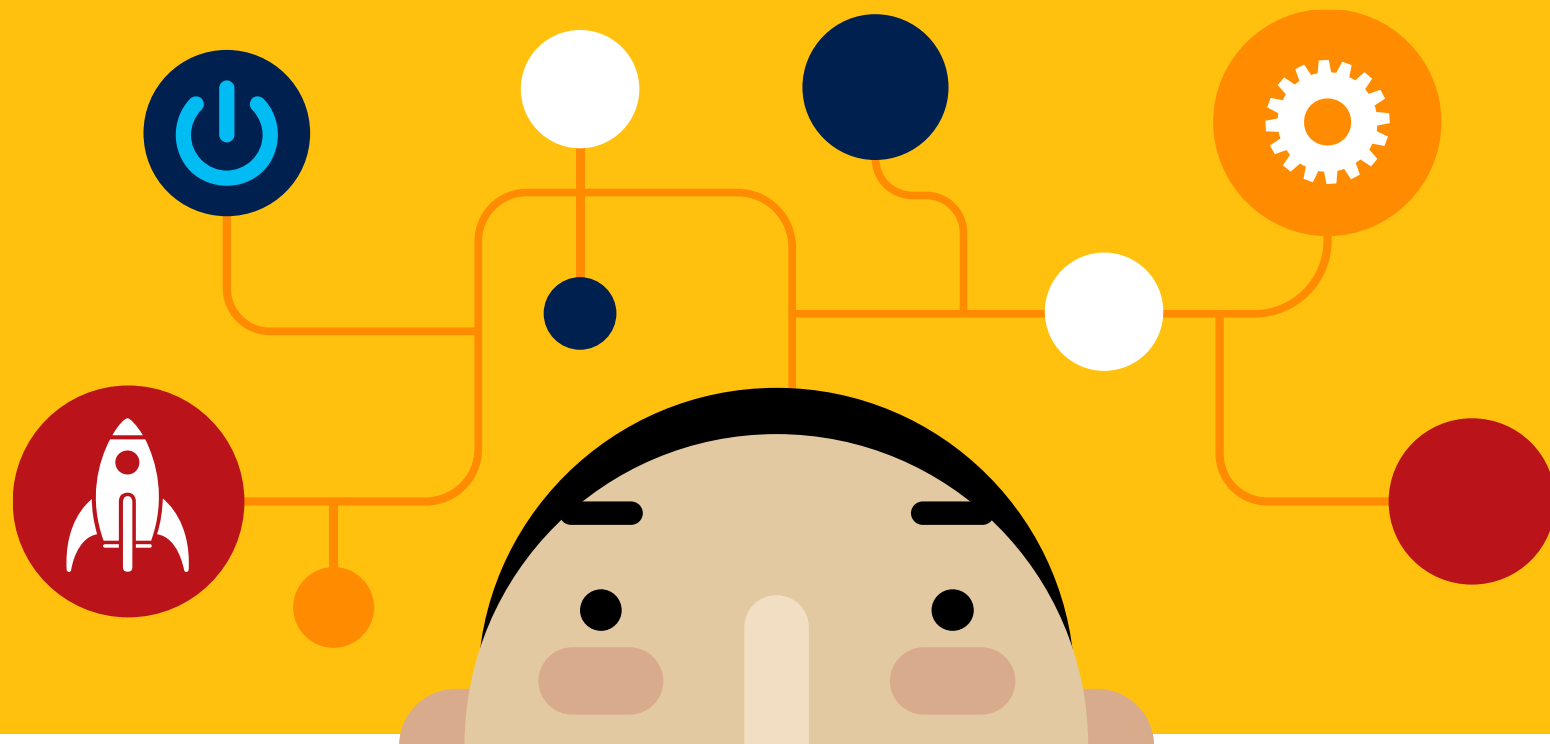


# React Componentes



# Saiba Mais

- Ao usar React com TypeScript, observe as dicas de tipagem em  
<https://react.dev/learn/typescript>  
<https://github.com/typescript-cheatsheets/react>  
<https://react-typescript-cheatsheet.netlify.app/>



# React - Componentes

- Uma interface de usuário com React é uma composição de components
- Um componente é formado por:
  - Marcações (HTML + CSS) que definem a aparência
  - Código JavaScript que define a lógica de funcionamento

# React - Componentes

☐ Only show products in stock

**Name Price**

**Fruits**

Apple \$1

Dragonfruit \$1

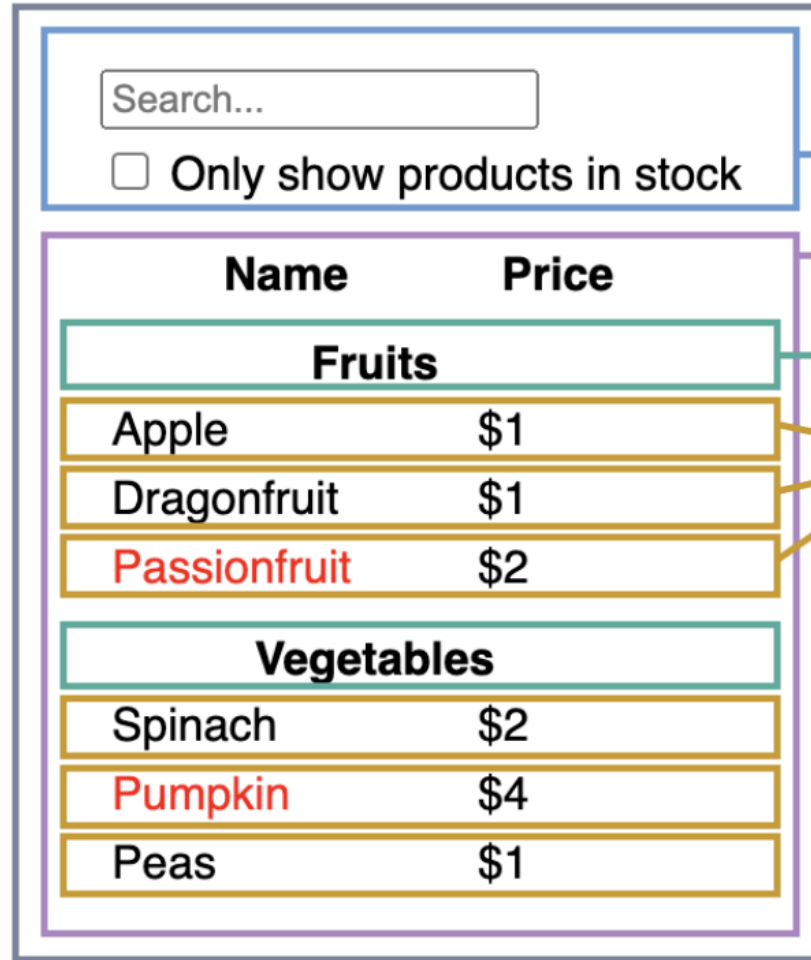
Passionfruit \$2

**Vegetables**

Spinach \$2

Pumpkin \$4

Peas \$1



1. FilterableProductTable

2. SearchBar

3. ProductTable

4. ProductCategoryRow

5. ProductRow

# React - Componentes


- Componentes === Máquina de Estados
  - React utiliza o modelo mental de máquina de estados para a IU
  - Atualiza-se o estado de um componente e o React renderiza a IU com base nesse estado
  - React possui controles internos para atualizar o DOM de forma eficiente
- Componentes === Funções
  - Componentes podem ser criados com base em funções (modelo mais atual)
  - Existe um suporte legado ao modelo com base em classes

# React - Componentes

```
import React from "react";

// Class components use ES6 class syntax
// Function components are now considered standard
class ClassComponent extends React.Component {
  render() {
    return <div> Hello World!</div>;
  }
}
```

Classe

 Função

```
import React from "react";

// React components are written as functions
function FunctionalComponentSyntax(props) {
  return <div>Hello World!</div>;
}
```

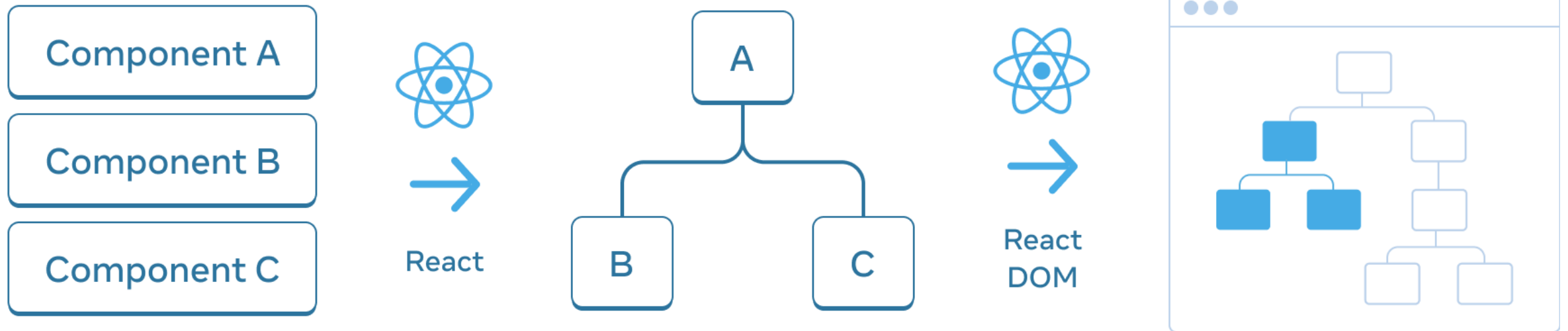
# React - Componentes

- Chamamos de “renderização” o processo de chamar um componente funcional do React
- Quando um componente é renderizado, ele retorna uma árvore contendo descrições de componentes que serão eventualmente traduzidos para nodos na árvore DOM no navegador
- O método **render()** é responsável por criar a árvore de componentes, inicializar a biblioteca React e anexar o nodos DOM gerados como filhos de um nodo DOM pai

```
function MyApp() {  
  return (<h1>Hello, world! </h1>);  
}  
  
const container = document.getElementById('root');  
const root = ReactDOM.createRoot(container);  
root.render(<MyApp />);
```



# React - Componentes



# React - JSX

- JSX é uma extensão sintática para JavaScript que permite escrever marcações no estilo do HTML dentro de um script de código
- Possui regras sintáticas mais restritas que o HTML

```
Sidebar() {  
  if (isLoggedIn()) {  
    <p>Welcome</p>  
  } else {  
    <Form />  
  }  
}
```

```
Form() {  
  onClick() {...}  
  onSubmit() {...}  
  
  <form onSubmit>  
    <input onClick />  
    <input onClick />  
  </form>  
}
```

# React - JSX

```
function MyButton() {  
  return (  
    <button>I'm a button</button>  
  );  
}
```

Sintaxe JSX  
<https://transform.tools/html-to-jsx>

```
export function MyApp() {  
  return (  
    <>  
      <h1>Welcome to my app</h1>  
      <MyButton />  
    </>  
  );  
}
```

- CUIDADO!
  - JSX utiliza letras maiúsculas para diferenciar entre elementos HTML e componentes React
  - Se a primeira letra for minúscula, assume que a tag é HTML, e se for maiúscula, assume ser uma variável em escopo que contém o componente React

# React - JSX

- JSX suporta interpolação de expressões JavaScript via `{ }`
  - Permite avaliar valores de variáveis, chamadas de funções, expressões condicionais ternárias, etc

```
function Greeting({ name }) {  
  return (  
    <h1>Hello, {name}</h1>  
  );  
}
```

```
export default function App() {  
  return (  
    <Greeting name="world" />  
  );  
}
```

# React - Props

- Props são valores passados de pai para filho
- Props são combinados em um único objeto somente de leitura/imutável
  - Componentes funcionais recebem *props* como argumento de entrada da função que define o componente
- Qualquer coisa pode ser passada como *prop*:
  - Valores primitivos, objetos, arrays, funções, tipos de componentes, elementos JSX, etc

# React - Props

```
interface Props {  
  name: string;  
}  
  
function HelloFunctionComponent(props: Props) {  
  return (  
    <div>  
      Hello {props.name}  
    </div>  
  );  
}  
  
export default HelloFunctionComponent;
```

# React - Renderização Condicional

```
let content;  
if (isLoggedIn) {  
  content = <AdminPanel />;  
} else {  
  content = <LoginForm />;  
}  
return (  
  <div>  
    {content}  
  </div>  
);
```

```
<div>  
  {isLoggedIn ? (  
    <AdminPanel />  
  ) : (  
    <LoginForm />  
  )}  
</div>
```

```
<div>  
  {isLoggedIn && <AdminPanel />}  
</div>
```

# React - Listas

- Para renderizar listas, o React necessita da inclusão do atributo *key* sobre cada elemento da lista
  - Valor deve ser uma *string* ou *number* que identifica unicamente o elemento da coleção
  - API do React utiliza esse valor para identificar quando novos elementos são inseridos ou removidos da coleção, ou então a ordem foi alterada

```
const products = [  
  { title: 'Cabbage', id: 1 },  
  { title: 'Garlic', id: 2 },  
  { title: 'Apple', id: 3 },  
];
```

```
const listItems = products.map(product =>  
  <li key={product.id}>  
    {product.title}  
  </li>  
);  
return (  
  <ul>{listItems}</ul>  
);
```



# React - Eventos

- React gerencia os eventos internamente
- Programador desenvolve tratadores de eventos e passa referências para esses tratadores como *props* para os componentes
  - Cuidado: dentro do JSX passamos a referência para a função e não uma string!
- Nomes de eventos são "camelCased": **onClick**, **onMouseOver**, etc
- Para prevenir a execução do comportamento default do evento DOM no React devemos chamar o método **event.preventDefault()** explicitamente dentro do tratador de evento
- Eventos suportados: <https://react.dev/learn/responding-to-events>

# React - Eventos

```
import React from 'react';
import './App.css';

function App() {
  const handleClick = () => {
    alert('Hello World');
  };

  const handleClick2 = (event: React.MouseEvent<HTMLButtonElement>) => {
    alert('Clicked: ' + event.currentTarget.name);
  };

  return (
    <button name='hello' onClick={handleClick}>
      Click Me!
    </button>
  );
}

export default App;
```