

Enunciado: Criando um Contador Dinâmico com React

Objetivo: Desenvolver um contador dinâmico que permita ao usuário incrementar, decrementar e resetar o valor. Este exercício ajudará a entender conceitos básicos de estados e eventos em React.

Passo 1: Criar o projeto React

1. Abra o terminal e execute o comando para criar um novo projeto React:

```
bash
Copiar código
npx create-react-app contador-react
```

2. Navegue até a pasta do projeto:

```
bash
Copiar código
cd contador-react
```

Passo 2: Estrutura do Componente

1. Dentro do diretório `src`, abra o arquivo `App.js`.
2. Limpe o conteúdo do arquivo, deixando apenas a estrutura básica:

```
import React from 'react';

function App() {
  return (
    <div className="App">
      <h1>Contador React</h1>
    </div>
  );
}

export default App;
```

Passo 3: Criar o Estado do Contador

1. Vamos utilizar o hook `useState` para gerenciar o valor do contador.
2. Importe o `useState` e crie um estado para o contador:

```
import React, { useState } from 'react';

function App() {
  const [count, setCount] = useState(0);

  return (
    <div className="App">
      <h1>Contador React</h1>
      <p>Valor atual: {count}</p>
    </div>
  );
}

export default App;
```

Passo 4: Adicionar Botões de Controle

1. Adicione três botões para incrementar, decrementar e resetar o contador.
2. Crie funções para cada ação e conecte-as aos botões:

```
import React, { useState } from 'react';

function App() {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(0);

  return (
    <div className="App">
      <h1>Contador React</h1>
      <p>Valor atual: {count}</p>
      <button onClick={increment}>Incrementar</button>
      <button onClick={decrement}>Decrementar</button>
      <button onClick={reset}>Resetar</button>
    </div>
  );
}

export default App;
```

Passo 5: Estilizando o Contador

1. Adicione um pouco de estilo para deixar o contador visualmente mais atraente. Abra o arquivo App.css e adicione o seguinte:

```
.App {
  text-align: center;
  margin-top: 50px;
}

button {
  margin: 5px;
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
}

p {
  font-size: 24px;
  font-weight: bold;
}
```

Passo 6: Testando a Aplicação

1. Execute o projeto para ver o contador em ação:

```
npm start
```

2. Interaja com os botões para verificar se o contador está funcionando conforme esperado.

Conclusão

Com este exercício, você aprendeu a criar e gerenciar estados em React, além de manipular eventos para controlar as ações do usuário. A partir daqui, você pode expandir a aplicação adicionando funcionalidades como limite máximo e mínimo para o contador, cores dinâmicas baseadas no valor, ou até mesmo guardar o estado em local storage para persistência.

Enunciado: Criando um Jogo da Velha com React

Objetivo: Desenvolver um jogo da velha interativo utilizando React. Este exercício ajudará a entender conceitos fundamentais como componentes, estados, props e eventos em React.

Passo 1: Configuração Inicial

1. Criar o Projeto React:

- Abra o terminal e crie um novo projeto React:

```
npx create-react-app jogo-da-velha
```

- Navegue até o diretório do projeto:

```
cd jogo-da-velha
```

2. Limpar o Projeto:

- No arquivo `src/App.js`, remova o conteúdo padrão, deixando apenas a estrutura básica:

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

function App() {
  return (
    <div className="App">
      <h1>Jogo da Velha</h1>
    </div>
  );
}

export default App;
```

3. Estrutura Básica do Projeto:

- Vamos criar componentes para o tabuleiro e para as células do jogo.

Passo 2: Criar o Componente de Célula (Square)

1. Criar o Componente Square:

- No diretório `src`, crie um arquivo `Square.js`:

```
import React from 'react';
```

```
function Square({ value, onClick }) {  
  return (  
    <button className="square" onClick={onClick}>  
      {value}  
    </button>  
  );  
}  
  
export default Square;
```

- Esse componente recebe duas props: `value`, que é o valor a ser exibido na célula, e `onClick`, que define o que acontece quando a célula é clicada.

2. Estilizar o Square:

- Adicione as seguintes regras de estilo no `App.css` para dar um visual agradável às células:

```
.square {  
  width: 60px;  
  height: 60px;  
  background-color: #fff;  
  border: 1px solid #000;  
  font-size: 24px;  
  font-weight: bold;  
  cursor: pointer;  
}
```

Passo 3: Criar o Componente de Tabuleiro (Board)

1. Criar o Componente Board:

- No diretório `src`, crie um arquivo `Board.js`:

```
import React, { useState } from 'react';  
import Square from './Square';  
  
function Board() {  
  const [squares, setSquares] = useState(Array(9).fill(null));  
  const [isXNext, setIsXNext] = useState(true);  
  
  const handleClick = (index) => {  
    const newSquares = squares.slice();  
    if (newSquares[index] || calculateWinner(squares)) return;  
  
    newSquares[index] = isXNext ? 'X' : 'O';  
    setSquares(newSquares);  
    setIsXNext(!isXNext);  
  };  
  
  const renderSquare = (index) => {  
    return   
      <Square value={squares[index]} onClick={ () =>  
        handleClick(index) } />;  
  };  
  
  const winner = calculateWinner(squares);  
  let status;  
  if (winner) {  
    status = `Vencedor: ${winner}`;  
  } else {  
    status = `Próximo jogador: ${isXNext ? 'X' : 'O'}`;  
  }  
}
```

```
    return (
      <div>
        <div className="status">{status}</div>
        <div className="board-row">
          {renderSquare(0)}
          {renderSquare(1)}
          {renderSquare(2)}
        </div>
        <div className="board-row">
          {renderSquare(3)}
          {renderSquare(4)}
          {renderSquare(5)}
        </div>
        <div className="board-row">
          {renderSquare(6)}
          {renderSquare(7)}
          {renderSquare(8)}
        </div>
      </div>
    );
  }

  function calculateWinner(squares) {
    const lines = [
      [0, 1, 2],
      [3, 4, 5],
      [6, 7, 8],
      [0, 3, 6],
      [1, 4, 7],
      [2, 5, 8],
      [0, 4, 8],
      [2, 4, 6],
    ];
    for (let i = 0; i < lines.length; i++) {
      const [a, b, c] = lines[i];
      if (squares[a] && squares[a] === squares[b] && squares[a] ===
squares[c]) {
        return squares[a];
      }
    }
    return null;
  }

  export default Board;
```

- o Este componente gerencia o estado do tabuleiro (squares) e determina qual jogador é o próximo (isXNext). A função calculateWinner verifica se há um vencedor.

2. Adicionar Estilos ao Tabuleiro:

- o No App.css, adicione os estilos do tabuleiro:

```
.board-row {
  display: flex;
}

.status {
  margin-bottom: 20px;
  font-size: 20px;
```

```
    font-weight: bold;
  }
```

Passo 4: Integrar os Componentes na Aplicação Principal

1. Atualizar o `App.js`:

- Agora, vamos integrar o componente `Board` no `App.js`:

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

function App() {
  return (
    <div className="App">
      <h1>Jogo da Velha</h1>
      <Board />
    </div>
  );
}

export default App;
```

2. Rodar a Aplicação:

- No terminal, execute o comando para iniciar o projeto:

```
npm start
```

- Acesse `http://localhost:3000` no navegador para ver o jogo em ação.

Passo 5: Melhorias e Reflexões

1. Adicionar um Botão de Reinício:

- Você pode adicionar um botão para reiniciar o jogo no componente `'Board'`:

```
<button onClick={() => setSquares(Array(9).fill(null))}>Reiniciar Jogo</button>
```

2. Expandir a Funcionalidade:

- Como desafio adicional, tente implementar:
 - Mensagem de empate quando todos os quadrados estiverem preenchidos e não houver vencedor.
 - Um histórico de jogadas para permitir que os jogadores voltem para movimentos anteriores.

Conclusão

Com este projeto, você aprendeu a criar um jogo da velha simples usando React, lidando com estados, props, e eventos. Esse exercício proporciona uma compreensão sólida de como construir aplicações interativas com componentes reutilizáveis em React.