# Componentes de classe

FUTURE4

# Sumário e Big Picture

# Componentes + Classes

- Vejam como é bom conhecer vários conceitos:
- Vocês conhecem componentes;
- Vocês conhecem classes;
- Logo, eu posso fazer:

Componente + Classe =

# Componentes + Classes

- Por que eu faria isso?
  - Pense em um *Post*, fica mais fácil de trabalhar com ele se, em um lugar só, eu puder controlar como ele se comporta e aparenta.
  - Nos dá acesso à propriedades especiais de componentes, algumas que veremos hoje e outras mais pra frente

# Componentes + Classes

- Como eu crio um componente de classe?

```
import React from 'react';
import PropTypes from 'prop-types';

class MeuComponente extends React.Component {
  constructor(props){
    super(props);
  }

  render(){
    return (
      <div>
          <h1>Oi {this.props.nome}! Eu sou um componente de classe.</h1>
      </div>
    )
  }
}

MeuComponente.propTypes = {
    nome: PropTypes.string.isRequired,
}
```

Pontos de atenção:

- *extends*
- *super*
- render()
- uso do *this*

# Componentes + Classes

- Como eu instancio um componente de classe?

```
import { MeuComponente } from "./MeuComponente"

const MeuApp = (props) => {
  return(
    <MeuComponente nome="Soter Padua" />
  )
}
```

Pontos de atenção:

- **Não usamos** a keyword *new*
- Uso **idêntico** ao de componentes funcionais

# Componente de Classe vs. Componente funcional

- Retomando, quais são as diferenças?

Classe:

```
class MeuComponente extends React.Component {
  constructor(props){
    super(props);
  }

  render(){
    return(
      <div>
          <h1>Oi {this.props.nome}! Eu sou um componente de classe.</h1>
      </div>
    )
  }
}

MeuComponente.propTypes = {
    nome: PropTypes.string.isRequired,
}
```

Funcional:

```
const MeuComponente = (props) => {
  return(
    <div>
        <h1>Oi {props.nome}! Eu sou um componente de classe.</h1>
    </div>
  )
}

MeuComponente.propTypes = {
    nome: PropTypes.string.isRequired,
}
```

# Beleza Soter, mas pra quê?

# Pausa: 5 min 

FUTURE4

# Estado interno

# Estado interno - Conceito

- Até agora fizemos componentes “burros”, que não conseguem interagir com informações passadas para eles, só recebem as props externamente (e props são read-only!).

- Queremos que os componentes tenham uma “memória” mesmo que dure só até a página atualizar.

- Introdução à **Reatividade**

FUTURE4

# Estado interno - Conceito

- **Imutabilidade** → **NUNCA** mudar valores do estado diretamente.

- Sempre que atualizamos algum valor no estado temos que deixar claro pro **React**, fazemos isso ao utilizar o método **setState**

# Estado interno - Exemplo

Exemplo no React:

Criação do estado com as propriedades e seus valores iniciais

Atualização do estado, passando um objeto inteiro que será o novo estado

Associação da função que atualiza o estado ao evento do click do botão

```
import React from 'react';  8.78 kB (gzip: 3.5 kB)

class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      numeroDeVezesClicado: 0,
    };
  }

  handleInputChange = () => {
    const numeroAtual = this.state.numeroDeVezesClicado;
    this.setState( state: { numeroDeVezesClicado: numeroAtual + 1 })
  };

  render() {
    return(
      <div>
        <h3>Número de vezes clicado: {this.state.numeroDeVezesClicado}</h3>
        <hr/>
        <button onClick={this.handleInputChange}>Clique em mim para aumentar o número de vezes clicado!</button>
      </div>
    )
  }
}

export default App;
```

# Estado interno - Demo

Demo aqui

FUTURE4

# Estado interno - Quando usamos?

- Pense que o **estado interno** de um componente é normalmente usado para coisas que só dizem respeito ao componente em si.

- Normalmente é usado para lidar com interações do usuário.

# Inputs Controlados - Conceito

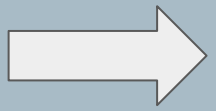
- Epa, mas inputs do HTML já tem um estado próprio
- Podemos usar o React para controlar o valor atual de campo, fazendo dele um campo “sincronizado” com o estado.
- Ao fazer isso temos de fácil acesso o valor atualizado de todos os *inputs*

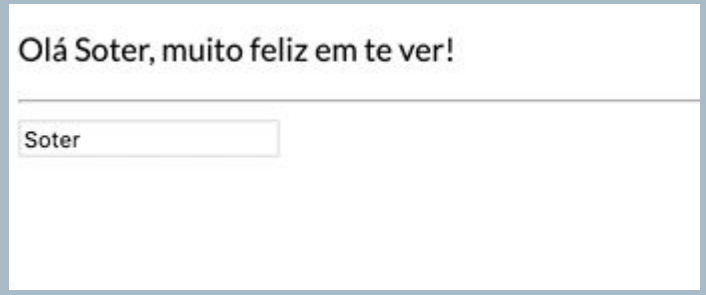
# Inputs Controlados - Exemplo

```
class App extends Component {
  constructor() {
    super();
    this.state = {
      name: "Soter"
    };
  }

  handleOnChangeName = (event) => {
    this.setState({ name: event.target.value });
  };

  render() {
    return (
      <div>
        <p>Olá {this.state.name}, muito feliz em te ver!</p>
        <hr />
        <input
          type="text"
          value={this.state.name}
          onChange={this.handleOnChangeName}
        />
      </div>
    );
  }
}
```






FUTURE4

# Pausa: 10 min 

FUTURE4

# Coding Together

FUTURE4

# Exercício 1

- Crie 3 *inputs* e 3 elementos de cabeçalho (h1, h2 e h3). Vincule o conteúdo do input com o do cabeçalho de tamanho correspondente.

# Exercício 2

- Vamos agora criar um botão que ao ser clicado mostra uma tag **h1**, e ao ser clicado novamente a esconde.

# Exercício 3

- Crie um novo botão que, ao ser clicado, deve limpar o valor de todos os campos.

# Exercício 4

- Crie uma imagem e um input do tipo *checkbox*, quando o input estiver ligado você deve mostrar uma imagem e quando estiver desligado, outra.

# Pausa: 5 min 

FUTURE4

# Review

FUTURE4

# Review

- Componentes + Classes = Mind Blown
- Estado Interno
- Inputs Controlados

# Dúvidas?

FUTURE4

# Obrigado!

FUTURE4