

React

<Conceitos>

Linguagens Script @ LEI / LEI-PL / LEI-CE

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

Conceitos Principais

- › Componentes
 - › Componentes Funcionais
 - › Componentes de Class
- › JSX
- › Props e State
- › Ciclo de Vida

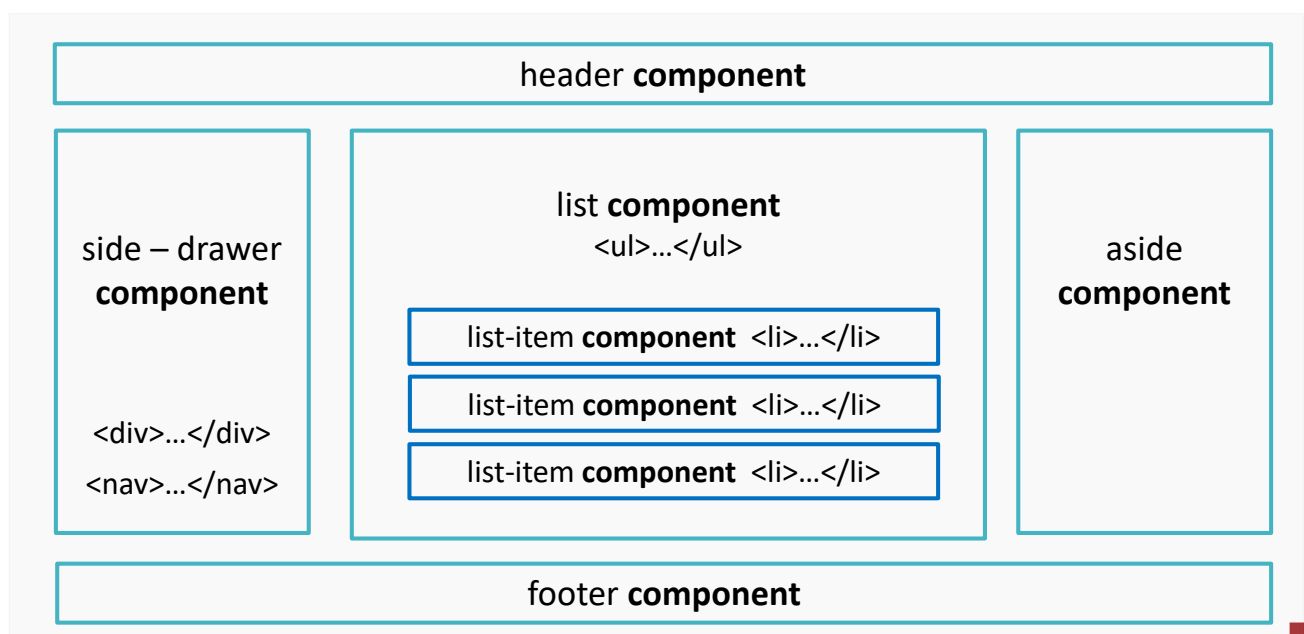
> React > Componentes

- O interface com o utilizador é composto por componentes (*React Components*)
 - Cada componente pode ser decomposto em outros componentes (arquitetura baseada numa estrutura hierárquica de *Components*)
- Os componentes permitem utilizar a mesma estrutura com outros dados (reutilização)



> React > Componentes

UI



> React > Componentes

- Componentes permitem que uma interface com o utilizador seja decomposta num conjunto de elementos reutilizáveis
 - cada um desses elementos pode ser considerado/aplicado de forma independente
- Componentes (à semelhança de uma função no JS):
 - aceitam inputs (**props**)
 - retornam o que deve ser renderizado no viewport do browser.
- Componentes podem ser definidos de duas formas distintas originando dois tipos de componentes:

> React > Componentes

- *Stateless Functional Component*
 - Componente simples, sem estado, que se trata apenas de uma função que obtém dados via props e retorna algum JSX que será renderizado na página.
- *Class-based components*
 - Componentes criados com `class`
 - Cada componente de class tem o método `render`, o qual descreve como o componente será renderizado na página;

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```
 - *Statefull Component* pois controla como o estado altera e a implementação da lógica do componente. Além disso, tem acesso a todas as diferentes partes do método do ciclo de vida do React

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```
- *Statefull Functional Component with Hooks*

> Componentes > Class Component

■ Class Component

- declarado com base em class e faz o extends do React.Component o que lhe permite ter acesso aos seus métodos
- O método render() é obrigatório e faz o return dos conteúdos definidos no componente através de código JSX para o ReactDOM fazer a ligação ao DOM (através dos métodos do DOM).
- O seu nome inicia-se sempre por maiúscula
- Não pode retornar 2 *siblings elements* (tem que existir sempre um wrapper)

```
class Site extends React.Component{
  render(){
    ...
    return(
      <div style={cardStyle}>
        <Header {...this.props}/>
        <Sidebar {...this.props}/>
        <Content {...this.props}/>
        <Footer {...this.props}/>
      </div>
    )
  }
}
```



> Componentes > Functional Component

■ Functional Component

- Uma função que retorna **JSX**.
- Tanto pode ser definida com base na declaração com base em function() ou em alternativa numa arrow function
- O seu nome inicia-se sempre por maiúscula
- Não pode retornar 2 *siblings elements* (tem que existir sempre um wrapper)

```
const Person = () => {

  return <h1> Function Component </h1>

}
```



Todos os componentes React devem agir como funções puras em relação a seus props.



< 49 >

> React Components

React

	Functional Component	Class Component
<i>Permite estados dentro do componente *</i>	✗	✓
<i>Usa métodos do ciclo de vida dentro do componente *</i>	✗	✓
<i>Usa atributo ref no component *</i>	✗	✓
<i>Usa hooks dentro do componente</i>	✓	✗

** Situação alterada com a introdução dos Hooks*

> React Components

■ Functional Components vs Class Components

Class Component	Functional Component (Using Hooks)
<i>componentDidMount</i> <i>componentDidUpdate</i> <i>componentWillUnmount</i>	<i>useEffect</i>
<i>state</i>	<i>useState</i>
<i>ref={contentRef => ref}</i>	<i>useRef</i>



> Class Component

```
<div id="container"></div>

<script type="text/babel">

  class HelloWorld extends React.Component{
    render(){

      return <p>Hello first component </p>
    }
  }

  ReactDOM.render(
    <HelloWorld/>, document.querySelector('#container')
  )
</script>
```

Cria um componente designado por HelloWorld. É um componente criado com base numa class que faz o extend de um React.Component

Todos os componentes React têm um método render() que geram os elementos HTML no browser através de JSX

Call the component <HelloWorld/>



> React Components > props

- À semelhança de uma função JS em que os argumentos são passados à função, no React as **props** são passados aos Componentes.
- Para que este fluxo de dados seja possível, é necessário percorrer 2 passos:
 - 1) Alterar a declaração do Componente
 - 2) Modificar a chamada do Componente

▪ 1) Alterar a declaração do Componente

```
class HelloWorld extends React.Component{  
  render(){  
  
    return <p>Hello, {this.props.nametarget} </p>  
  }  
}
```

O acesso ao valor da propriedade é feito através de:

this.props.property

O JSX utiliza chavetas {...} para avaliar uma expressão



> React Component

▪ 2) Modificar a chamada do Componente

O valor da prop é passado definindo um atributo com o mesmo nome da propriedade (property) definida na declaração do componente

```
ReactDOM.render(  
  <div>  
    <HelloWorld nametarget='Coimbra!'>/>  
    <HelloWorld nametarget='Porto!'>/>  
    <HelloWorld nametarget='Lisboa!'>/>  
  </div>,  
  document.querySelector('#container')  
)
```

Importante: Num componente, não existe limite para o número de atributos destinados à passagem de props



> React Component

```
<body>
<div id="container"></div>

<script type="text/babel">

  class HelloWorld extends React.Component{
    render(){

      return <p>Hello, {this.props.nametarget} </p>

    }
  }

  ReactDOM.render(
    <div>
      <HelloWorld nametarget='Coimbra!'/>
      <HelloWorld nametarget='Porto!'/>
      <HelloWorld nametarget='Lisboa!'/>
    </div>,
    document.querySelector('#container')
  )
</script>
</body>
```

← → ↺
Apps eP

Hello, Coimbra!
Hello, Porto!
Hello, Lisboa!

Um valor passado como prop a um componente não deve ser modificado (*immutable data*).

Garante-se assim que qualquer componente que referencia essa prop recebe o mesmo valor do que os outros componentes que referenciam essa mesma prop.



> Props

```
const Header = (props) => {

  var headerStyle={
    backgroundColor:'orange',
    height:'10vh',
  }

  return (
    <div style={headerStyle}>
      <h1>{props.cheader}</h1>
    </div>
  )

}
```

...



JSX



< 57 >

> JSX

React

- JSX é uma biblioteca Javascript que permite simplificar a criação de elementos
 - Muito semelhante ao HTML mas não é HTML
 - O JSX necessita de ser convertido (*transpiling*) paJavaScript (plain JS) de forma a alterar o HTML.
 - Utiliza uma sintaxe muito mais amigável e mais simples, mas uma vez convertido o código a criação dos elementos é feita com base na função `React.createElement()`
 - A conversão de JSX para plain JS é assegurado pelo Babel
 - O código JSX pode:
 - ser definido / armazenado numa variável em qualquer lugar no código JavaScript
 - mais vulgarmente, no `render()/return` dos componentes quer sejam definidos como *functional components* ou *class components*

> JSX

- JSX é convertido (nos bastidores) em JavaScript simples que pode ser reconhecido pelo navegador
 - JSX para JavaScript simples
 - JSX é mais eficiente e muito mais simples.

```
class App extends Component{
```

```
  render(){
```

```
    return(
```

```
      //<div className="App">
```

```
      //  <h1>First React Component</h1>
```

```
      // </div>
```

Códigos são equivalentes

```
      React.createElement('div',{className: 'App'},React.createElement('h1',null,'First React Component'))
```

```
    );
```

```
  }
```

```
}
```

```
export default App;
```

First React Component



> JSX

Olá, LScript!

```
function JsxExemplo() {  
  return React.createElement("h1",{},{}, "Olá, LScript!")  
}
```

```
function JsxExemplo() {  
  return <h1>Olá,LScript!</h1>  
}
```

```
const JsxExemplo = () => <h1>Olá, LScript!</h1>;
```

```
const JsxExemplo = () => React.createElement("h1", {},  
"Olá, LScript!")
```



> JSX

- Permite:

- *opening e closing tags* `<opening tag> **** </closing tag>`
- *self closing tag* `<tag property="value" />`
- para **representar elementos HTML** as *tags* são definidas em minúsculas
- as *tags* dos componentes são definidas em **maiúsculas**

```
class ButtonChild extends React.Component{
  render(){
    return <button type='this.props.behaviour'> {this.props.children} </button>
  }
}

ReactDOM.render(
  <div>
    <ButtonChild behaviour='submit'>SEND DATA!</ButtonChild>
  </div>,
  document.querySelector('#container')
)
```

> JSX

- Retornar múltiplos elementos:

```
class Display extends React.Component{
  render(){
    return (
      <p>p1</p>
      <p>p2</p>
      <p>p3</p>
    )
  }
}
```

```
Uncaught SyntaxError: Inline Babel
script: Adjacent JSX elements
must be wrapped in an enclosing
tag (11:24)
   9 |
  10 |
    <p>p1</p>
    > 11 |
      <p>p2</p>
        |
      ^
    12 |
    <p>p3</p>
    13 |
    14 |
```

Não pode retornar 2
siblings elements
(tem que existir
sempre um wrapper)

- Solução 1: Definir um container (muito comum)

```
class Display extends React.Component{
  render(){
    return (
      <div>
        <p>p1</p>
        <p>p2</p>
        <p>p3</p>
      </div>
    )
  }
}
```

p1

p2

p3

O eventual problema
desta abordagem é
que cria sempre um
elemento HTML

■ Retornar múltiplos elementos:

- Solução 2: retornar um array no qual todo o markup é definido como elementos individuais do array

```
class Display extends React.Component{
  render(){
    return (
      [
        <p>p1</p>,
        <p>p2</p>,
        <p>p3</p>
      ]
    )
  }
}
```

p1
p2
p3

- Solução 3: **<React.Fragment>** ou **<> </>** que permite criar um wrapper sem necessidade de definir um array e com a vantagem de não criar markup adicional

```
class Display extends React.Component{
  render(){
    return (
      <React.Fragment>
        <p>p1</p>
        <p>p2</p>
        <p>p3</p>
      </React.Fragment>
    )
  }
}
```

p1
p2
p3



■ Regras de Capitalização:

- Se o JSX for armazenado numa variável esta deve ser declarada em minúsculas

```
const caprules = <h1> Capitalization Rules</h1>

class Display extends React.Component{
  render(){
    return (
      <div>{caprules}</div>
    )
  }
}
```

Capitalization Rules

- Um componente definido como *function component* deve ser em maiúsculas

```
const Display = () =>{
  return (
    <h1> Capitalization Rules</h1>
  )
}
```

- Um componente definido como *class component* deve ser em maiúsculas

```
class Display extends React.Component {
  render(){
    return (
      <h1> Capitalization Rules</h1>
    )
  }
}
```



> JSX

- `{}`
 - As chavetas são um **elemento absolutamente essencial** em JSX porque permitem avaliar código *plain JavaScript* (janela JS no código JSX)
 - Indicam ao transpiler (Babel) que o seu conteúdo é uma expressão JavaScript a ser executada:
 - Valores das variáveis e objetos
 - Chamadas de funções e Métodos
 - Expressões Condicionais

> JSX

- ```
const team = "Milwaukee Bucks";
const user = {first:"Giannis",
 last:"Antetokounmpo"}

const getFullName = (user) => {
 return `${user.first} ${user.last}`
}
const Welcome = () => {

 return (
 <div>
 <h1>{getFullName(user)} welcome to {team}!</h1>
 </div>)
 }

 ReactDOM.render(
 <div>
 <Welcome/>
 </div>,
 destination)
```

Chamada a uma função      variável

**Giannis Antetokounmpo welcome to Milwaukee Bucks!**

## > JSX

- {}

- Expressões Condicionais em JSX

- a condição é avaliada diretamente no JSX (operador ternário, se verdadeiro retorna o 1º argumento caso contrario retorna o 2º)

```
const Welcome = () => {
 const isLoggedIn = false;
 return <div> {
 isLoggedIn ? (
 <h1> Welcome!</h1>): (<h1>Please login!</h1>) }
 </div>
 }
}
```

Expressão  
Condicional

```
ReactDOM.render(
 <div>
 <Welcome/>
 </div>,
 destination
)
```

**Please login!**



## > JSX

- {}

- Expressões Condicionais em JSX

- Apesar de possível a abordagem anterior, é muito comum definir a condição fora, armazenar o seu output numa variável e depois aceder ao seu valor no JSX

```
const Welcome = () => {
 const isLoggedIn = false;
 const welcomeMessage = isLoggedIn ? (
 <h1> Welcome!</h1>): (<h1>Please login!</h1>);
 return <div> {welcomeMessage} </div>
}
```

Expressão  
Condicional

```
ReactDOM.render(
 <div>
 <Welcome/>
 </div>,
 destination
)
```

**Please login!**



## > JSX

- {}

- A possibilidade de aceder ao valor de uma variável permite inserir formatações diretamente no JSX (posteriormente serão abordadas formas mais abrangentes de formatar os conteúdos)

- Ao atributo style é atribuído um JS object (Style Object) declarado fora do JSX

```
class Label extends React.Component{
 render(){
 var labelStyle = {
 fontFamily:"sans-serif",
 fontWeight:"bold",
 padding:13,
 margin:0
 }
 return(
 <p style={labelStyle}>{this.props.color}</p>
)
 }
}
```

As propriedades CSS só com uma palavra mantêm-se, as propriedades definidas em CSS com hífen passam a camelCase (ex: fontFamily)



## > JSX

- Inserir comentários

- É possível inserir comentários em JSX de duas formas distintas
  - Se o comentário é filho de uma tag deve ser envolvido em chavetas

```
ReactDOM.render(
 <div className="slideIn">
 <p className="emphasis">Gabagool!</p>
 {/* I am a child comment */}
 <Label/>
 </div>,
 document.querySelector("#container")
);
```



## > JSX

- Inserir comentários

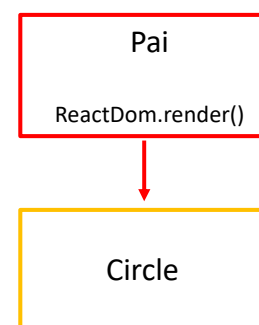
- O JSX permite uma outra forma de inserir comentários, inseridos na opening tag e tendo por base exclusivamente a sintaxe do JS:

- `/* .... */`      múltiplas linhas
- ```
ReactDOM.render(  
  <div className="slideIn">  
    <p className="emphasis">Gabagool!</p>  
    <Label  
      /* This comment  
       goes across  
       multiple lines */  
      className="colorCard" // end of line  
    />  
  </div>,  
  document.querySelector("#container")  
>);
```

> JSX

- A forma mais comum de definir código JSX passa pela sua implementação no interior do render() do componente:

```
class Circle extends React.Component{  
  render(){  
    var circleStyle = {  
      padding: 10,  
      margin: 20,  
      display: "inline-block",  
      backgroundColor: "#F9C240",  
      borderRadius: "50%",  
      width: 100,  
      height: 100};  
    return(  
      <div style={circleStyle}></div>  
    )  
  }  
};  
  
ReactDOM.render(  
  <div> <Circle/> </div>,  
  destination  
)
```



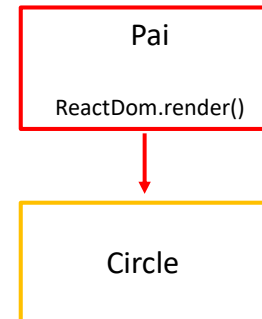
> JSX

- No entanto essa forma pode ser um pouco restritiva, por vezes pode ser vantajoso armazenar o código JSX:

- Numa variável e depois no método render() aceder ao valor dessa variável.

```
class Circle extends React.Component{
  render(){
    var circleStyle = {
      padding: 10,
      margin: 20,
      display: "inline-block",
      backgroundColor:this.props.bgcolor,
      borderRadius: "50%",
      width: 100,
      height: 100,
    };
    return(
      <div style={circleStyle}></div>
    )
  }
};

const jsxalt = <Circle bgcolor="lightgreen"/>
ReactDOM.render(
  <div> {jsxalt} </div>,
  destination
)
```



> JSX

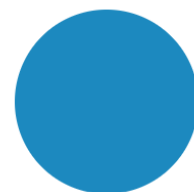
- Numa função, p.ex. neste caso para gerar a cor de forma aleatória:

```
class Circle extends React.Component{
  render(){
    var circleStyle = {
      ...
      backgroundColor:this.props.bgColor,
      ...};
    return(
      <div style={circleStyle}></div>
    )
  }
};

function showCircle(){
  var colors= ["#393E41", "#E94F37", "#1C89BF", "#A1D363"];
  var ran = Math.floor(Math.random() * colors.length)
  console.log(ran);console.log(colors[ran])

  return <Circle bgColor={colors[ran]} />
}

ReactDOM.render(
  <div> {showCircle()}</div>,
  destination)
```



2
#1C89BF



1
#E94F37

- Tirar partido da versatilidade dos arrays para gerar JSX de forma automática:

```
class Circle extends React.Component{
  render(){
    var circleStyle = {
      ...
      backgroundColor:this.props.bgColor,
      ...};
    return(
      <div style={circleStyle}></div>
    )
  }
};
```



```
const colors = ["#393E41", "#E94F37", "#1C89BF", "#A1D363",
               "#85FFC7", "#297373", "#FF8552", "#A40E4C"];
let renderData=[];

for (var i = 0; i < colors.length; i++) {
  let color = colors[i];
  renderData.push(<Circle key={i + color} bgColor={color} />); }

ReactDOM.render(
  <div> {renderData} </div>,
  destination)
```