

**Aplicações para a Internet II**

NPM, ReactJS para Node

2021/2022

# <React/>



- 
- Na definição dada pelos seus criadores, React é “uma biblioteca JavaScript declarativa, eficiente e flexível para a criação de user interfaces (UI)”.
  - É uma biblioteca de **JavaScript**, open source, usada para construir user interfaces nomeadamente para aplicações de página única.
  - Tem como principal objetivo, ser rápida, escalável e simples, podendo ainda ser usada em combinação com outras bibliotecas ou frameworks de JavaScript.
  - Foi primeiramente implementada no feed do Facebook no ano de 2011, tendo sido criada por **Jordan Walke**, um engenheiro de software que trabalhava para esta empresa.

# <React/>



- 
- A biblioteca React.js é usada para lidar com a camada de visualização para **aplicações web e mobile**, e permite também criar componentes de UI reutilizáveis.
  - Dá a possibilidade, aos developers, de criarem aplicações web onde possam alterar elementos ou os dados exibidos, sem recarregar a página. Um exemplo simples são os **likes do Facebook** onde o número de likes pode aumentar ou diminuir sem ter de se fazer “refresh” na página.

# <Requisitos/>



- Utilizar React na construção das nossas aplicações pode ser feito de duas formas diferentes, diretamente no HTML ou Utilizando o NPM e Node.js.
- Após a instalação do node.js é possível usar o comando npm (create-react-app) para instalar o componente que permite fazer a criação automática de aplicações react.

*npm install -g create-react-app*

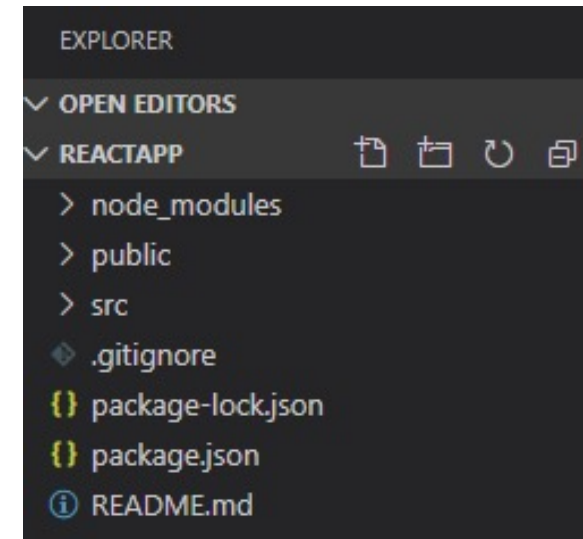
- Criar a aplicação React:

*Create-react-app <nome da aplicação>*

# <Estrutura/>



- A aplicação React cria automaticamente as pastas necessárias, como mostra na figura:



- .gitignore – É o ficheiro que permite identificar a origem GIT para controlar quais são os ficheiros e pastas que serão ignorados quando é feito commit ao código. Embora o comando create-react-app, crie o ficheiro, o mesmo processo não cria as pastas no repositório GIT.

# <Estrutura/>



- .package.json – ficheiro que contém as dependências e os scripts necessários para o projeto.

```
1 {
2   "name": "reactapp",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^4.2.4",
7     "@testing-library/react": "^9.4.0",
8     "@testing-library/user-event": "^7.2.1",
9     "react": "^16.12.0",
10    "react-dom": "^16.12.0",
11    "react-scripts": "3.3.1"
12  },
13  "scripts": {
14    "start": "react-scripts start",
15    "build": "react-scripts build",
16    "test": "react-scripts test",
17    "eject": "react-scripts eject"
18  },
19  "eslintConfig": {
20    "extends": "react-app"
21  },
22  "browserslist": {
23    "production": [
24      ">0.2%",
25      "not dead",
26      "not op_mini all"
27    ],
28    "development": [
29      "last 1 chrome version",
30      "last 1 firefox version",
31      "last 1 safari version"
32    ]
33  }
34 }
35 |
```

# <Estrutura/>



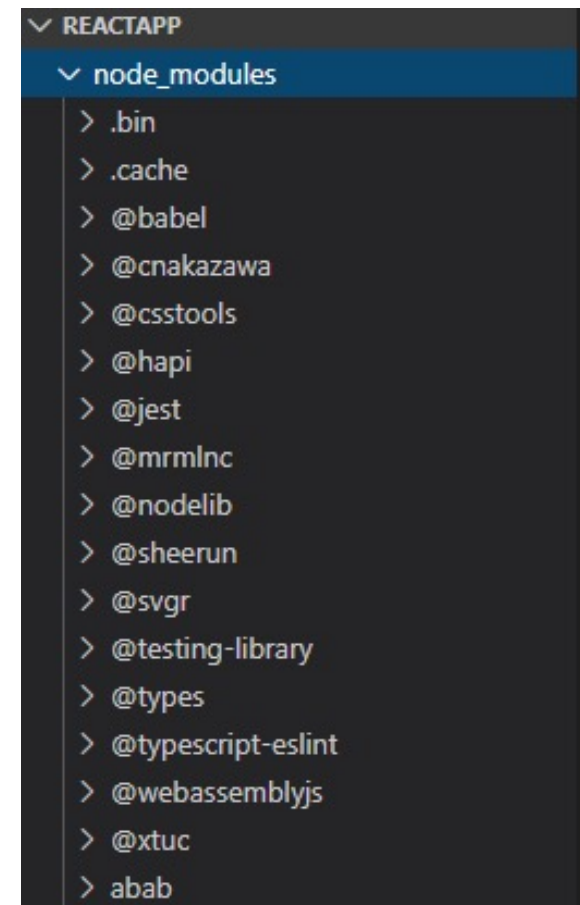
- .package-lock.json contém a árvore de dependência exata a ser instalada em /node\_modules. Mantém um histórico de alterações feitas no package.json.

```
1 {
2   "name": "reactapp",
3   "version": "0.1.0",
4   "lockfileVersion": 1,
5   "requires": true,
6   "dependencies": {
7     "@babel/code-frame": {
8       "version": "7.8.3",
9       "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.8.3.tgz",
10      "integrity": "sha512-a9gxpmdXtZEInkCSHUJDLHZVBgb1QS0jhss4cPP93EW7s+uC5bikET2twEF3KV+7rDb1JcmNvTR7VJeqd2C2g==",
11      "requires": {
12        "@babel/highlight": "^7.8.3"
13      }
14    },
15     "@babel/compat-data": {
16       "version": "7.8.5",
17       "resolved": "https://registry.npmjs.org/@babel/compat-data/-/compat-data-7.8.5.tgz",
18       "integrity": "sha512-jWYuQX/Ob0hG1UiEkBH5SANS8/8oKXiQWjj7p7xgj9Zmnt//aUvyz4dBkk0HNS8/cbyC5NmmH87VekW+mXFg==",
19       "requires": {
20         "browserslist": "^4.8.5",
21         "invariant": "^2.2.4",
22         "semver": "^5.5.0"
23       }
24     },
25     "dependencies": {
26       "semver": {
27         "version": "5.7.1",
28         "resolved": "https://registry.npmjs.org/semver/-/semver-5.7.1.tgz",
29         "integrity": "sha512-sauauDzU0SWroHbQXw0QwIoQKwzQhuoCjS1ogOi02/h4krB5pZ8C/xw/Ioy4rTTKxvg3MQdAoU7bu23HyqKhPg=="
30       }
31     }
32   }
33 }
```

# <Estrutura/>



- `.node_modules` - A pasta contém todas as dependências e subdependências especificadas no ficheiro `package.json` e usadas pela aplicação React.
- Contém mais 800 subpastas, sendo que essa pasta é adicionada automaticamente no ficheiro `.gitignore`.

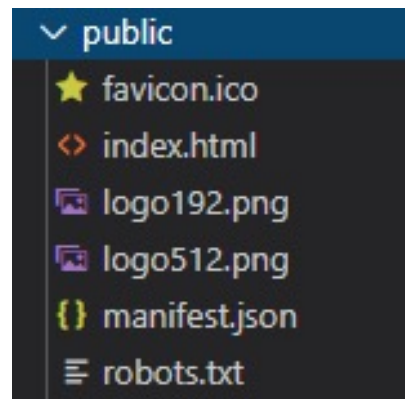




# <Estrutura/>



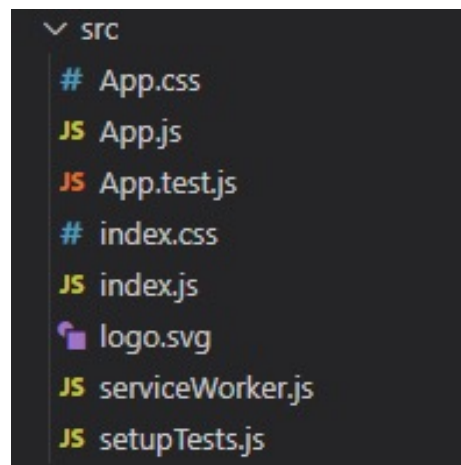
- .public - Esta pasta contém ficheiros que não exigem processamento adicional pelo webpack. O arquivo index.html é considerado como um ponto de entrada para a web application.



# <Estrutura/>



- .src - Esta pasta é pasta principal das aplicações React. Contém JavaScript que precisa ser processado pelo webpack.
- Nesta pasta, existe o componente principal (App.js), juntamente com o seu estilo (App.css), suíte de testes (App.test.js), Index.js e index.css.
- Estes ficheiros fornecem um ponto de entrada para a aplicação.



# <Componente/>



- 
- Em React, um componente é referido como um ‘pedaço’ de código isolado que pode ser reutilizado em vários módulos.
  - A aplicação React contém um componente raiz na qual outros subcomponentes estão incluídos, por exemplo, numa página, a aplicação é subdividida em 3 partes, o Header, o Main e o Footer.
  - Assim sendo, existe um único componente com 3 subcomponentes.
  - Existem 2 tipos de componentes no React.js
    - Componente funcional sem estado (Stateless Functional Component)
    - Componente de classe com estado (Stateful Class Component)

## Stateless Functional Component

- Este tipo de componente inclui funções JavaScript simples.
- Esses componentes incluem propriedades imutáveis, ou seja, o valor das propriedades não pode ser alterado.
- Um componente funcional é usado principalmente para a user interface.

```
const StatelessComp = props => {  
  return (  
    <div> props.description </div>  
  )  
};
```

# <Componente/>



## Stateful Class Component

- As classes component são as classes que estendem a classe Component da biblioteca React.
- As classes component devem incluir o método render que retorna HTML.

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
  render() {  
    return (  
      <div>  
        <h1>Hello world </h1>  
        <h2>{this.state.date.toLocaleTimeString()} </h2>  
      </div> );  
    }  
  }  
}
```

# <Props e State/>



- 
- Os componentes precisam de dados para trabalhar, como Props ou State.
  - Props ou State determinam o que é que o componente processa e como se comporta internamente.

# <Props/>



- Se os componentes forem funções javascript o Props será o input da função.
- Nesta analogia, um componente aceita um input (Props) processa a informação e renderiza código JSX.



- Props recebe dados do componente pai em modo read only.

# <Props/>



- 
- Os dados no Props, embora sejam acessíveis a um componente, a filosofia do React é que o Props deverá ser imutável e de cima para baixo.
  - Significa que um componente pai pode transmitir dados para os seus filhos como Props, mas os componentes filho não o podem modificar.

## Importante

- A classe / função importada deve estar em maiúsculas, caso contrário dará um erro na consola.
- As classes e componentes retornam código JSX, ou seja Javascript xml.



# <State/>



- É um objecto que pertence ao componente onde é declarado. O seu âmbito está limitado ao componente atual.
- Um componente pode iniciar o seu estado e atualizá-lo sempre que necessário. O estado do componente pai normalmente termina como sendo Props do componente filho. Quando o estado é ultrapassado do âmbito atual, passa a ser tratado como Props.



- State é usado internamente para comunicação dentro do próprio componente.

# <JSX/>



- 
- JSX é um formato JavaScript XML usado em aplicações React, embora não exclusivo, mas torna mais fácil a criação de aplicações em React.
  - Torna o código mais legível, confiável e fácil de modificar. JSX é uma extensão para JavaScript.
  - O JSX usa a sintaxe HTML para criar elementos e componentes.
  - JSX compila o código em JavaScript puro, sendo assim entendido pelos browsers.
  - O JSX permite escrever elementos HTML em JavaScript e colocá-los na DOM sem recorrer aos métodos `createElement()` e `/` ou `appendChild()`.

# <JSX/>



- JSX não é interpretado pelo browser, React usa o Babel para interpretar e transformar em código JavaScript tradicional.

```
<h1 className="title">Hello World</h1>
```



```
React.createElement(  
  "h1",  
  { className: "title" },  
  "Hello World"  
);
```

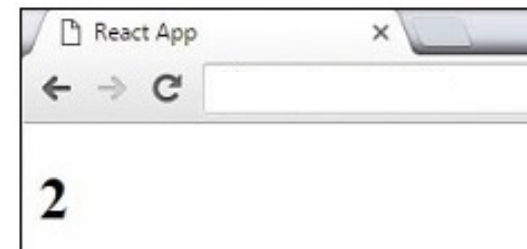
# <JSX/>



- Expressões JavaScript podem ser usadas dentro do JSX. Para isso é necessário colocar o código entre {}. Ex:

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

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>{1+1}</h1>  
      </div>  
    );  
  }  
}  
export default App;
```



# <JSX/>



- Não podemos usar instruções if else dentro do JSX, mas podemos usar expressões condicionais. Ex:

```
import React from 'react';

class App extends React.Component {
  render() {
    var i = 1;
    return (
      <div>
        <h1>{i == 1 ? 'True!' : 'False'}</h1>
      </div>
    );
  }
}
export default App;
```



# <JSX/>



- O JSX foi criado para ser parecido com HTML, mas com o poder de criação e utilização de componentes reutilizáveis. Ao retornar JSX de uma função, podemos criar essa reutilização:

```
function User(props) {  
  return <div>User: {props.name} is {props.age}</div>  
}  
  
function App() {  
  return (  
    <div>  
      { /* Dois elementos USER */ }  
      <User name="João" age="34" />  
      <User name="Abel" age="25" />  
    </div>  
  )  
}
```

# <JSX/>



- O Babel “transforma” (compila e traduz) o elemento <User /> em:

```
React.createElement(User, { name: “Abel”, age: “25” })
```

- Podemos verificar como o Props é convertido num objeto. Este objeto é passado como primeiro parâmetro da função User.
- No cenário anterior, é passado um Props em string, contudo é possível enviar dados com vários tipos:

```
<User name=“Julie” age={25} />
```

- O Babel “transcreve” o elemento em:

```
React.createElement(User, { name: “Julie”,  
age: 25 })
```

# <JSX Classname/>



- **className** substitui **class**, pois **class** é uma palavra reservada em JavaScript:

```
// JSX  
<div className="active">Hello</div>
```

```
// HTML  
<div class="active">Hello</div>
```



# <JSX Mapas e chaves/>



- Tendo em conta o seguinte exemplo:

```
function UserList() {  
  const users = ['João', 'Abel']  
  
  return (  
    <ul>  
      <li>{users[0]}</li>  
      <li>{users[1]}</li>  
    </ul>  
  )  
}
```


- Os dois elementos li vão ser filhos (Props children) para o elemento ul. React precisa que os filhos sejam interpretados e para isso, cria internamente uma matriz.

# <JSX Mapas e chaves/>



- A interpretação anterior pode ser imitada da seguinte forma.

```
function UserList() {  
  const users = ['João', 'Abel']  
  
  return (  
    <ul>  
      <li>{users[0]}</li>  
      <li>{users[1]}</li>  
    </ul>  
  )  
}
```



```
function UserList() {  
  const users = ['João', 'Abel']  
  
  return (  
    <ul>  
      {  
        <li>{users[0]}</li>  
        <li>{users[1]}</li>  
      }  
    </ul>  
  )  
}
```

- Embora não seja aconselhável, e a melhor opção é usar o JSX para o fazer, a demonstração anterior é possível e funciona.

# <JSX Mapas e chaves/>



- O código anterior não é dinâmico e por cada User, teria que ser alterado o código.
- Em vez de se codificar o array de Users, passa-se por props e utiliza-se a função **map** para iterar os valores:

```
function UserList(props) {  
  return (  
    <ul>  
      { props.users.map( name => {  
        return <li>{name}</li>  
      })}  
    </ul>  
  )  
}  
<UserList users={['Adolfo', 'Raimundo']} />
```

- A utilização de map para iteração é fundamental porque é necessário converter um array noutra array – um array de strings para um array de JSX.

# <JSX Mapas e chaves/>



- O React interpreta e constroi a DOM quando se desenvolve em JSX.
- O React faz esse processo de forma automática e transparente, exceto no caso de fornecermos ao JSX um array (quando utilizamos map).
- É necessário fornecer uma chave única para que o React possa rastrear os elementos que constrói.
- Map fornece o index que pode ser utilizado com chave no segundo argumento da função:

```
{props.users.map((name, index) => {  
    return <li  
key={index}>{name}</li>  
})}
```

# <JSX Mapas e chaves/>



- **Importante sobre as chaves:**

- A **Key** só precisa ser única dentro do array, não para toda a aplicação ou componente.
- As **Keys** podem ser string, desde que sejam únicas
- Usar as **Keys** como sendo os índices dos arrays, pode ser uma má ideia, ou então deve ser garantido que os arrays não sofrem alterações enquanto a aplicação estiver online.
- O Ideal será utilizar as **Keys** como sendo os ID's das bases de dados, pois mesmo que um registo seja eliminado o index do array muda, mas o da tabela não.