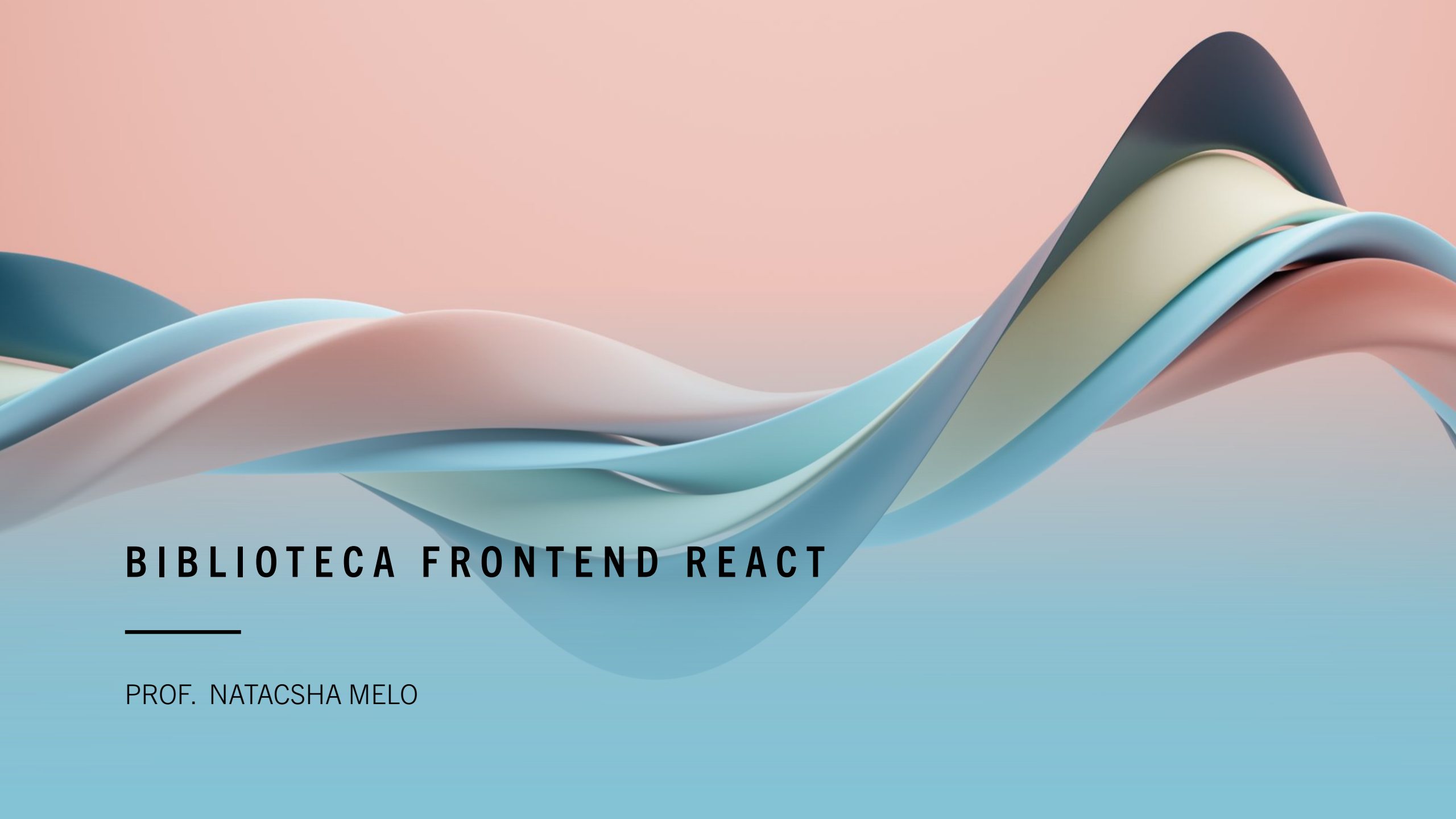


BIBLIOTECA FRONTEND REACT

PROF. NATACSHA MELO

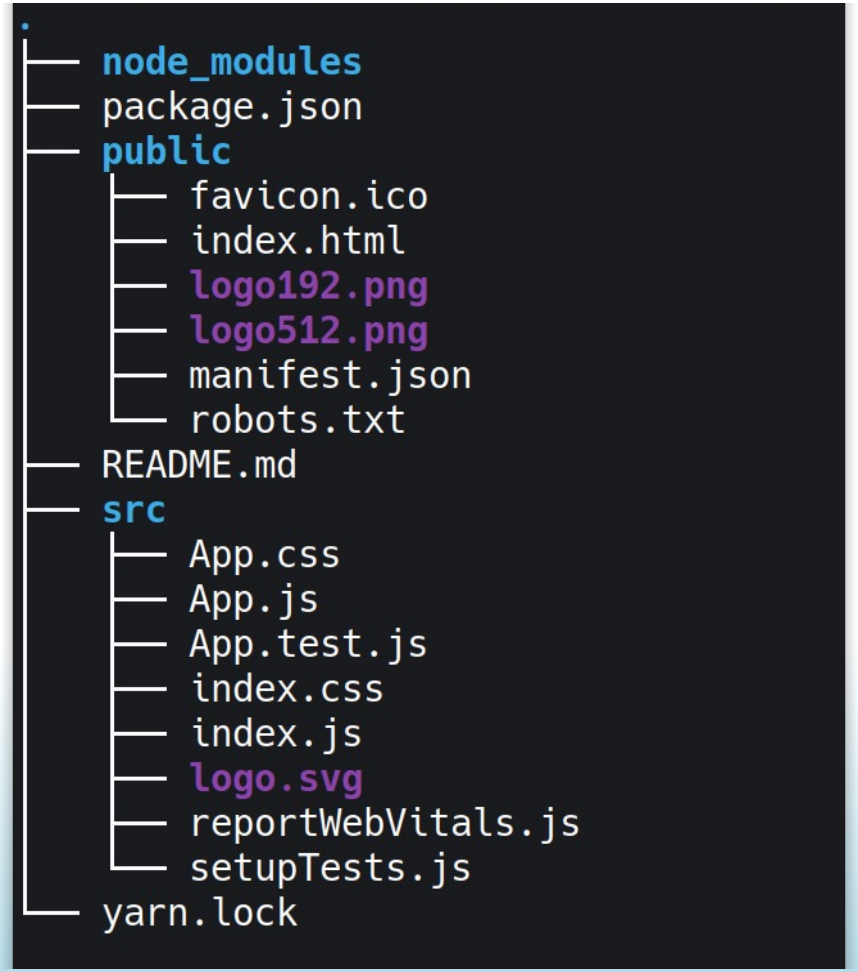
O PACOTE CREAT-REACT-APP

- Utilizar bibliotecas Javascript em uma página HTML não é eficiente
- O Creat React App cria um Webpack compilado com os arquivos que precisamos
- Para configurar create-react-app, execute os passos apresentados aqui.
- Para um inicio rápido execute os seguintes passos

```
npx create-react-app my-app
cd my-app
npm start
```

ESTRUTURA BÁSICA DE UMA APLICAÇÃO COM REACT

O **create-react-app** gera um projeto básico, contendo em sua raíz os arquivos *.gitinore, package.json, READM E.md, yarn.lock*, e as pastas *public* e *src*, onde o código da aplicação ficará.



CONFIGURANDO O PROJETO REACT-APP

- **/public** possui o arquivo importante **index.html** semelhante ao que fizemos
- O **/src** conterá todo o nosso código react
- Teste o ambiente para ver como compila editando a linha que contem o conteúdo "To get started, edit `src/App.js` and save to reload." no arquivo /src/App.js
- Exclua todos os arquivos do /src mantenho apenas o **index.css e index.js**
- No arquivo src/index.js importe o React, ReactDOM e o arquivo CSS

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
```

CONFIGURANDO O PROJETO REACT-APP

- Vamos criar nosso App Componente novamente. Utilizaremos **className** ao invés de **class** no src/index.js

```
class App extends React.Component {
    render() {
        return (
            <div className="App">
             <h1>Olá, React!</h1>
             </div>
        )
    }
}
```

- Por fim, para renderizar o App para o root adicionamos

```
ReactDOM.render(<App />, document.getElementById('root'))
```

- Copie e cole este arquivo no index.css

CONFIGURANDO O PROJETO REACT-APP

- Vamos criar nosso App Componente novamente. Utilizaremos **className** ao invés de **class** no src/index.js

```
class App extends React.Component {
    render() {
        return (
            <div className="App">
             <h1>Olá, React!</h1>
             </div>
        )
    }
}
```

- Por fim, para renderizar o App para o root adicionamos

```
ReactDOM.render(<App />, document.getElementById('root'))
```

<App /> é um componente, que será carregado no elemento **id=root** de **public/index.html**

- Copie e cole este arquivo no index.css

CONFIGURANDO O PROJETO REACT-APP

Aqui está o nosso arquivo index.js. Desta vez, estamos carregando o Componente como uma propriedade do React, então não precisamos mais estender React.Component.

```
import React, { Component } from 'react' import ReactDOM from 'react-dom’
import './index.css’

class App extends Component {
    render() {
        return (
                <div className="App">
                    <h1>Hello, React!</h1>
                </div>
                 )
    }
 }
ReactDOM.render(<App />, document.getElementById('root'))
```

VIRTUAL DOM E JSX

- No mundo do React, o termo “virtual DOM” é geralmente associado aos Elementos do React uma vez que eles são objetos representando a interface do usuário
- Essa abordagem permite a API declarativa do React onde você diz ao React qual o state que você quer que a interface do usuário esteja, e ele garante que o DOM seja igual ao state
- Quando se trata do código React ele não é exatamente um HTML. Ele é JSX, JavaScript XML
- Podemos criar e usar tags semelhantes a XML:

```
const heading = <h1 className="site-heading">Olá, React!</h1>
```

O JSX

- O JSX possui mais semelhanças sintáticas com o JavaScript do que com o HTML
- `className` é usado ao invés de `class` para adicionar classes CSS. Visto que `class` é uma palavra reservada em JavaScript
- Propriedades e métodos em JSX são camelCase: ao invés de `onchange` usa-se `onChange`
- Tags de fechamento automático devem terminar em uma barra: `<img />`
- As expressões JavaScript também podem ser incorporadas ao JSX usando chaves, incluindo variáveis, funções e propriedades:

```
const name = 'Natacsha'
const heading = <h1>Hello, {name}</h1>
```

ADIÇÃO DE ESTILOS

- No React, você especifica uma classe CSS com className
- Funciona da mesma forma que o class atributo HTML:

```
<img className="avatar" />
```

- Então você escreve as regras CSS para ele em um arquivo CSS separado:

```
/* In your CSS */

.avatar {

border-radius: 50%;

}
```

O React não prescreve como você adiciona arquivos CSS. Pode adicionar via <link> ou de qualquer outra forma.

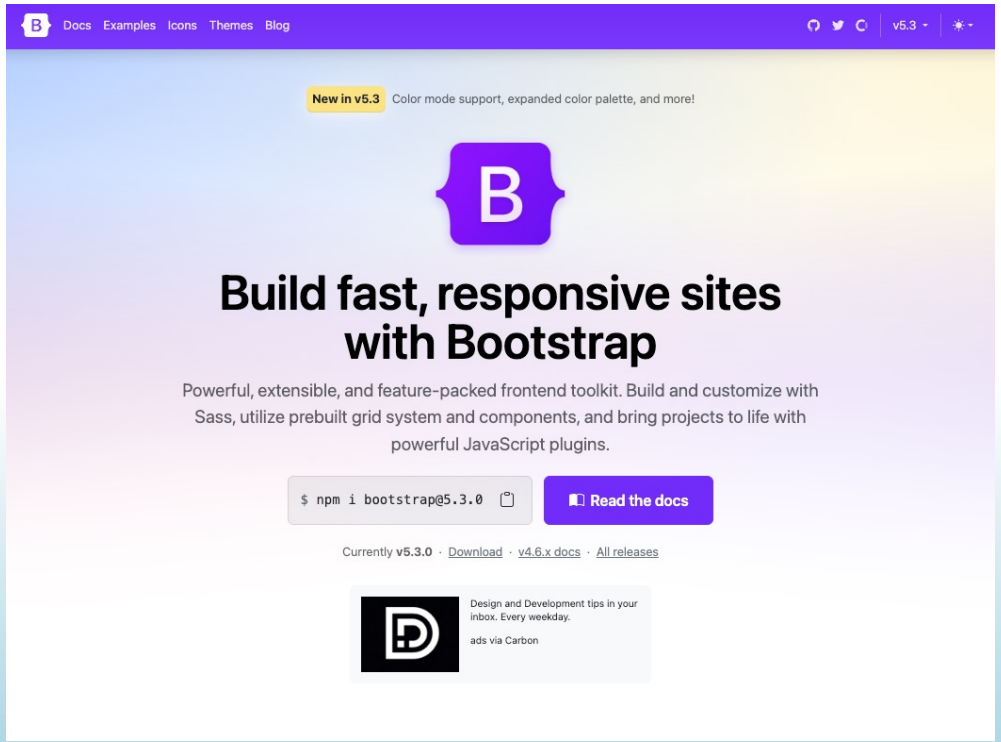
ADICIONANDO O BOOTSTRAP E O FONTAWESOME

- O framework Bootstrap será usado em nossa aplicação de exemplo, e para instalá-lo basta rodar o comando abaixo

```
$ npm install bootstrap
```

- Abra o arquivo src/index.js e adicione

```
import 'bootstrap/dist/css/bootstrap.css';
```



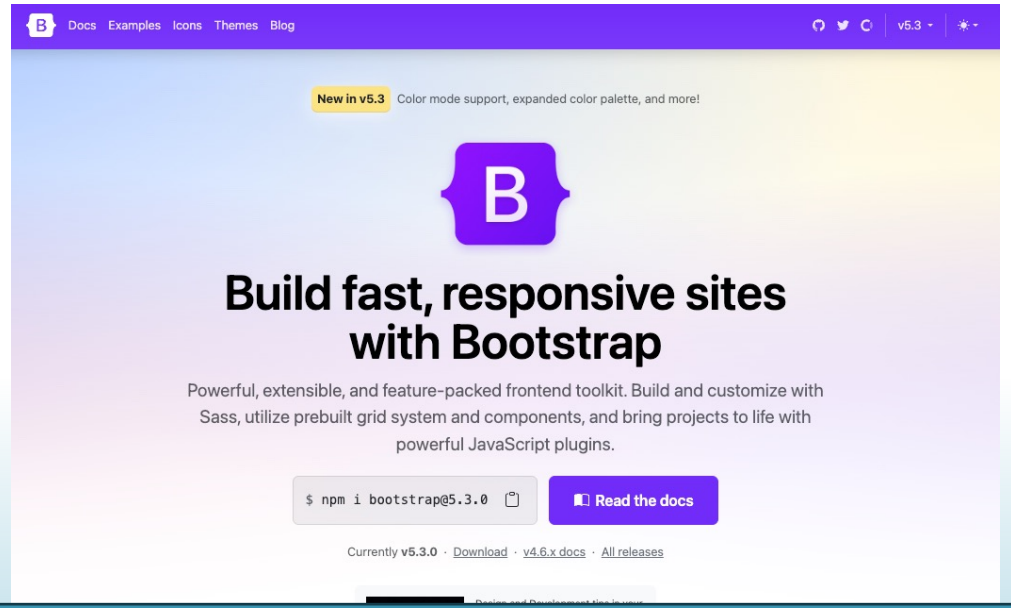
ADICIONANDO O BOOTSTRAP E O FONTAWESOME

- O framework Bootstrap será usado em nossa aplicação de exemplo, e para instalá-lo basta rodar o comando abaixo

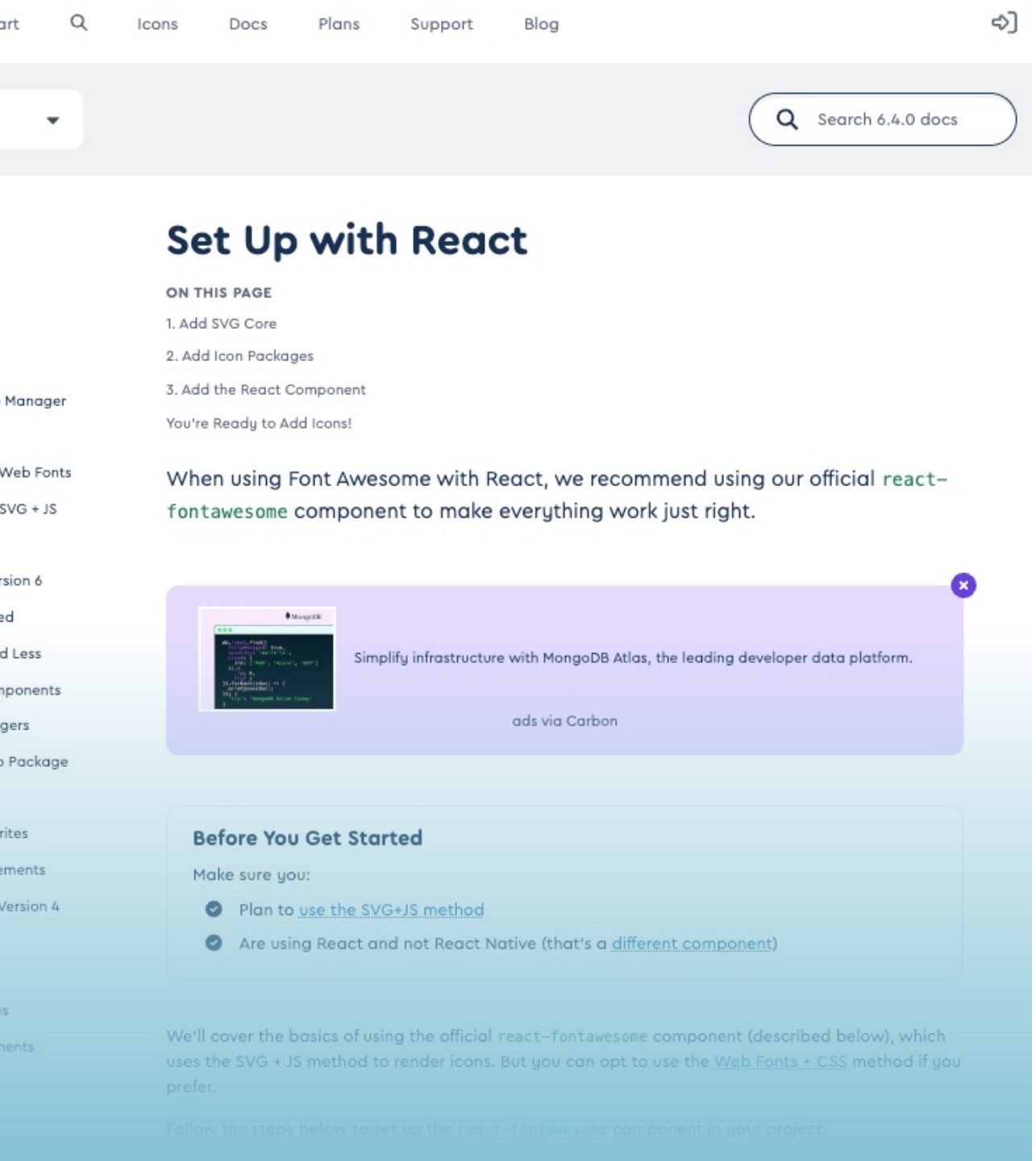
```
$ npm install bootstrap
```

- Abra o arquivo src/index.js e adicione

```
import 'bootstrap/dist/css/bootstrap.css';
```



Uma aplicação React é toda desenvolvida através de código ES6, e daí o uso do **import** ao invés de **require**



ADICIONANDO O BOOTSTRAP E O FONTAWESOME

- Adicione o SVG Core
- Adicione os pacotes de ícones
- Adicione o componente React

Siga as instruções aqui.

COMPONENTES

- A maioria dos aplicativos React têm muitos componentes pequenos
- Tudo é carregado no Componente App principal
- Os componentes possuem seu próprio arquivo
- Remova a Classe App do index.js
- Crie um arquivo App.js e adicione o código ao lado nele

```
import React, { Component } from 'react'

class App extends Component {
        render() {
                return (
                        <div className="App">

                        <h1>Hello, React!</h1>
                        </div>
                         )
        }
}
 export default App
```

COMPONENTES DE CLASSE

- Vamos criar um componente de tabela
- Crie um arquivo src/Table.js com o código ao lado
- O componente possui letra maiúscula para diferenciar dos elementos padrão do HTML
- Importe o componente no arquivo App.js

```
import Table from './Table'
```

- Carregue o Componente no render da Classe App
- Substitua o texto por <Table /> e altere a className para container

```
import React, { Component } from 'react'

class Table extends Component {
     render() {
          return (
               <table>
                    <thead>
                         <tr>
                              <th>Nome</th>
                              <th>e-mail</th>
                         </tr>
                    </thead>
                    <tbody>
                         <tr>
                              <td>Natacsha</td>
                              <td>nat@mail.com</td>
                         </tr>
                         <tr>
                              <td>João</td>
                              <td>joao@mail.com</td>
                         </tr>
                         <tr>
                              <td>José</td>
                              <td>jose@mail.com</td>
                         </tr>
                         <tr>
                              <td>Anna</td>
                              <td>anna@mail.com</td>
                         </tr>
                    </tbody>
               </table>
          )
     }
}

export default Table
```

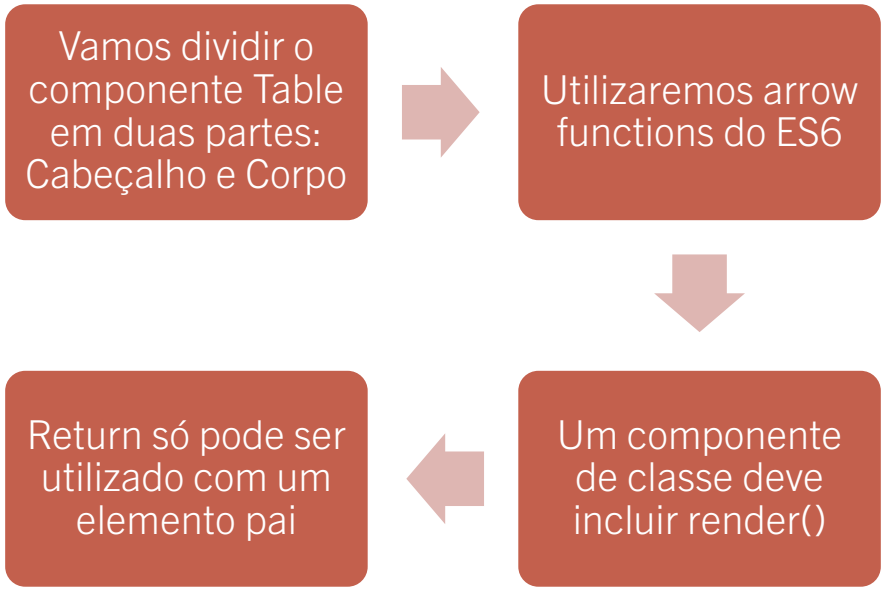
HIERARQUIA DE COMPONENTES

Para trabalhar com a hierarquia utilizaremos os chamados componentes simples

Utiliza formato de função

Não utiliza a palavra class

HIERARQUIA DE COMPONENTES




```
import React, { Component } from 'react'

const TableHeader = () => {
return (
<thead> ...</thead>
)
}

const TableBody = () => {
return (
<tbody>
...
</tbody>
)
}

class Table extends Component {
render() {
return (
<table className='table'>
<TableHeader />
<TableBody />
</table>
)
}
}

export default Table
```

COMPONENTE SIMPLES VERSUS CLASSE

```
const SimpleComponent = () =>
{
return <div>Example</div>
}
```

```
class ClassComponent extends Component
{
    render() {
        return <div>Example</div>
    }
}
```