



Front-end: React

React - Aula 04

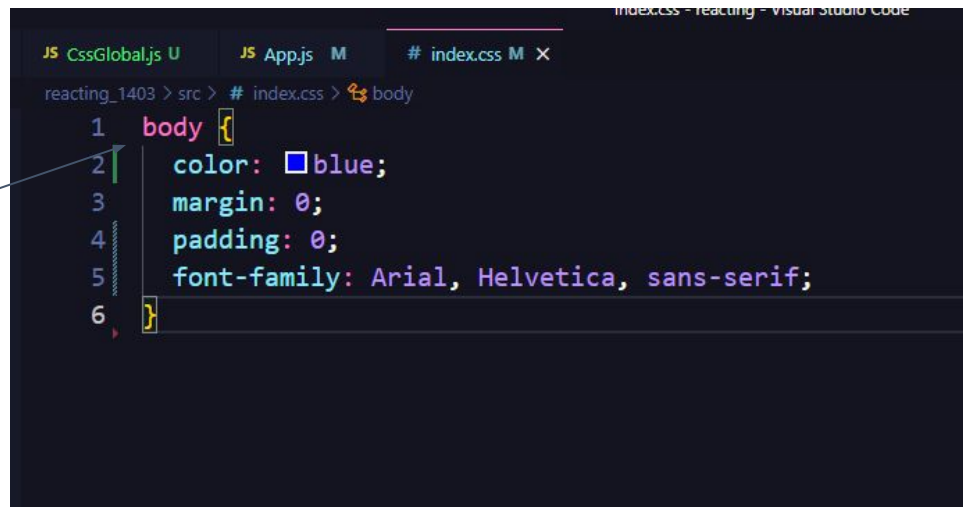
Prof. MSc. Kelson Almeida

Agenda

- React com CSS
 - CSS global
 - CSS de componentes
 - CSS inline
 - Classes dinâmicas
 - CSS Modules
- Formulários com React

CSS Global

- Estilizar elementos em comum ou fazer reset no CSS
- Index.css (src)
 - Os estilos globais vão estar nesse arquivo.



The screenshot shows a code editor with three tabs: 'JS CssGlobal.js U', 'JS App.js M', and '# index.css M X'. The active tab is '# index.css M X'. The editor content shows the following CSS code:

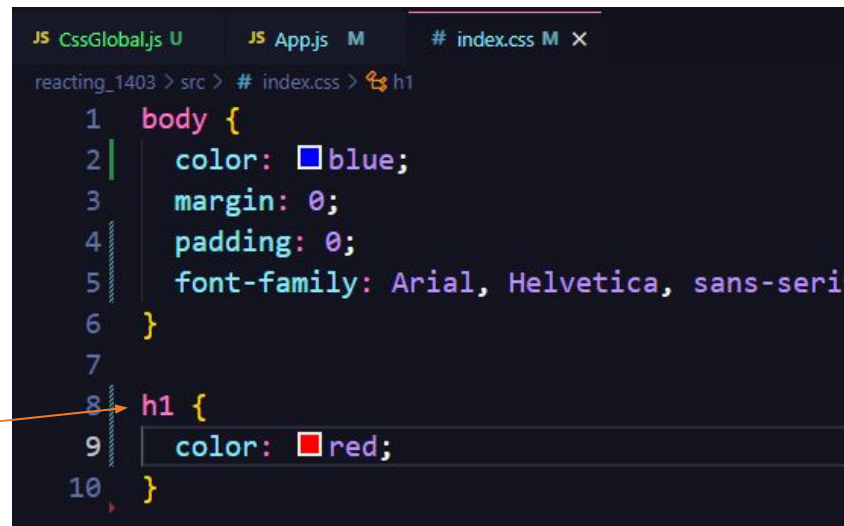
```
reacting_1403 > src > # index.css > body
1  body {
2    color: blue;
3    margin: 0;
4    padding: 0;
5    font-family: Arial, Helvetica, sans-serif;
6  }
```

A blue line points from the first bullet point of the list to the 'body' selector in the code.

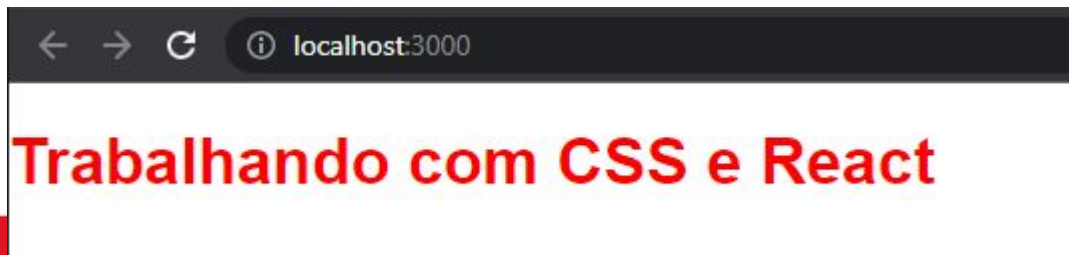


CSS Global

- Estilizar elementos em comum ou fazer reset no CSS
- Index.css (src)
 - Os estilos globais vão estar nesse arquivo.



```
JS CssGlobal.js U JS App.js M # index.css M x
reacting_1403 > src > # index.css > h1
1  body {
2    color: blue;
3    margin: 0;
4    padding: 0;
5    font-family: Arial, Helvetica, sans-serif;
6  }
7
8  h1 {
9    color: red;
10 }
```



Vamos praticar?

- Crie um estilo global que contenha:
 - A cor de fundo do corpo deve ser #d77d7d.
 - O tamanho da fonte padrão deve ser definido como 50px.
 - Todas as imagens devem ter uma borda sólida de 1px na cor #ccc.
- Crie um componente que mostre um parágrafo <p> e imagem para verificar se o CSS surtiu efeito.



CSS inline

- Inline do React é bem parecido com o do CSS
- Podemos encontrar em alguns projetos o atributo **style** diretamente em algum componente.
- É preferível que optemos por outra maneira, inline dificulta a manutenção.
- Detalhes:
 - `{{ }}` (duas chaves para abrir e fechar o style.
 - camelCase para separar as palavras comportas, no exemplo ao lado “background-color” virou “backgroundColor”

```
CSS

const minhaDiv = (
  <div
    style={{
      backgroundColor: 'blue',
      color: 'white'
    }}
  >
    Olá, mundo!
  </div>
);
```

Vamos praticar?

- Adicione um estilo **inline** usando CSS em um componente React.
- O estilo deve ter o seguinte:
 - A cor do texto deve ser vermelha (#ff0000).
 - O fundo deve ser azul (#0000ff).
 - A largura deve ser de 200 pixels.
 - O alinhamento do texto deve ser centralizado.



CSS Inline dinâmico

- Estilo baseado em uma condicional
- No exemplo ao lado temos um caso com o if ternário que aprendemos
- Dependendo do valor mudou a regra de exibição de um “background”

CSS

```
const isDarkMode = true;
const minhaDiv = (
  <div
    style={{
      backgroundColor: isDarkMode ? 'black' : 'white',
      color: isDarkMode ? 'white' : 'black'
    }}
  >
    Olá, mundo!
  </div>
);
```


CSS de Componente

- Para o componente em específico
- Mesmo nome do componente, importando no componente.
- É criado um exemplo com (App.jsx e App.css)
- O CSS pode vaziar para outros componentes.
 - Tem que ter cuidado!
 - Ele pode vaziar, devemos usar regras mais restritas.
 - Usar className da tag para fechar o escopo do CSS somente na tag do componente desejado.

```
JS CssComponente.js U x # CssComponente.css U x
reacting_1403 > src > components > AulasCss > .JS CssComponente.js > [C] CssComponente
1 import './CssComponente.css'
2
3 const CssComponente = () => {
4   return (
5     <div>
6       <p>Um parágrafo no CSS de Componente...</p>
7     </div>
8   )
9 }
10
11 export default CssComponente
```

```
JS CssComponente.js U x # CssComponente.css U x
reacting_1403 > src > components > AulasCss > # CssComponente.css > [C] p
1 p {
2   color: pink;
3   background-color: cornflowerblue;
4 }
```

← → ↻ ⓘ localhost:3000

Trabalhando com CSS e React

Um parágrafo no CSS de Componente...

CSS de Componente

- Para o componente em específico
- Mesmo nome do componente, importando no componente.
- É criado um exemplo com (App.jsx e App.css)
- O CSS pode vaziar para outros componentes.
 - Tem que ter cuidado!
 - Ele pode vaziar, devemos usar regras mais restritas.
 - Usar className da tag para fechar o escopo do CSS somente na tag do componente desejado.

```
JS CssComponente.js U x # CssComponente.css U
reacting_1403 > src > components > AulasCss > JS CssComponente.js > [0] CssComponente
1 import './CssComponente.css'
2
3 const CssComponente = () => {
4   return (
5     <div>
6       <p className="p-CssComponente">Um parágrafo no CSS de Componente...</p>
7     </div>
8   )
9 }
10
11 export default CssComponente
```

```
JS CssComponente.js U # CssComponente.css U x
reacting_1403 > src > components > AulasCss > # CssComponente.css > .p-CssComponente
1 p {
2   color: black;
3   background-color: cornflowerblue;
4 }
5
6 .p-CssComponente {
7   color: pink;
8   background-color: cornflowerblue;
9 }
```

localhost:3000

Trabalhando com CSS e React

Um parágrafo no CSS de Componente...

CSS dinâmico (com classes)

- Poderíamos fazer esse dinamismo sem a obrigatoriedade de ser em inline
- Mais interessante.
- Por exemplo, criando as classes CSS e chamando a classe a ser usada de acordo com uma condicional

CSS

```
.light-mode {  
  background-color: white;  
  color: black;  
}  
  
.dark-mode {  
  background-color: black;  
  color: white;  
}
```

javascript

```
import React from 'react';  
import './styles.css';  
  
const minhaDiv = ({ isDarkMode }) => {  
  const classe = isDarkMode ? 'dark-mode' : 'light-mode';  
  return <div className={classe}>Olá, mundo!</div>;  
};
```

CSS Module

- Arquivos CSS modulares.
- CSS que corresponde a cada componente.
- O estilo fica totalmente contido no escopo do componente.
- Evitando mais ainda vazar para outros componentes.
- O arquivo criado com o final “.module.css”

javascript

```
import React from 'react';
import styles from './MeuComponente.module.css';

const MeuComponente = () => {
  return (
    <div className={styles.myClass}>
      Olá, mundo!
    </div>
  );
};

export default MeuComponente;
```

CSS

```
.myClass {
  color: red;
}

.myOtherClass {
  background-color: blue;
}
```

Vamos praticar?

- Crie um CSS Modularizado para um componente chamado Campanha.
- Esse componente exibe na tela uma mensagem de acordo com o mês.
 - Essa frase deve ser exibida na cor preta.
- A cor de fundo de uma tarja (pode ser uma div) e a mensagem (dentro da tarja) devem mudar de acordo com o mês que passamos como prop (string) para o componente filho.
 - **Setembro** -> cor: amarelo, mensagem: Prevenção ao suicídio.
 - **Outubro** -> cor: rosa, mensagem: Conscientização sobre o câncer de mama.
 - **Novembro** -> cor: azul, mensagem: Prevenção e combate ao câncer de próstata.



Formulários

- Tag <form>
- Labels contém: htmlFor, com o valor do name do input
- O mais recomendado é utilizar a tag <label> ... </label> entre os inputs
- Não utiliza action, o processamento é assíncrono.

```
jsx

import React from 'react';

function Formulario() {
  return (
    <div>
      <form>
        <div>
          <label>
            Nome:
            <input type="text" name="nome" />
          </label>
        </div>
        <div>
          <label>
            E-mail:
            <input type="email" name="email" />
          </label>
        </div>
      </form>
    </div>
  );
}

export default Formulario;
```

Manipular valores de input

- Com o hook `useState`
- Armazenar na variável de estado e usar o “set” para mudar o valor.
- Criar uma função para alterar o valor no evento `onChange`.
- Pensando em: Enviar dados para um back-end -> banco de dados. Por exemplo.

```
1  import { useState } from 'react'
2
3  const ManipularValores = () => {
4
5      const [nome, setNome] = useState()
6      const [email, setEmail] = useState()
7
8      const handleName = (e) => {
9          setNome(e.target.value)
10     }
11
12     const handleEmail = (e) => {
13         setEmail(e.target.email)
14     }
15 }
```

```
16  return (
17      <div>
18          <form>
19              <label>
20                  Nome:
21                  <input name='nome' type='text' onChange={handleName} />
22              </label>
23              <br/>
24              <label>
25                  Email:
26                  <input name='email' type='text' onChange={handleEmail} />
27              </label>
28              <br/>
29              <button>Enviar</button>
30          </form>
31      </div>
32  )
33 }
```

Envio de formulário

- onSubmit dentro da tag form.
- Criamos uma função para receber o processamento do onSubmit, geralmente damos o nome de “handleSubmit”.

```
21 const handleSubmit = (e) => {
22   e.preventDefault()
23   console.Log("Vai enviar o seguinte formulário para o back-end: ")
24   console.Log(`Nome: ${nome} Email: ${email} Curso: ${curso}`)
25 }
26
27 return (
28   <div>
29     <h1>Cadastro de Aluno</h1>
30     <form onSubmit={handleSubmit}>
```

The screenshot shows a web browser at localhost:3000 with a form titled "Cadastro de Aluno". The form contains three input fields: "Nome:" with the value "Kelson", "Email:" with the value "kelson@uniesp.edu.br", and "Curso:" with the value "Sistemas para Internet". Below the inputs is an "Enviar" button. To the right, the browser's developer console is open, showing the following log messages:

```
Vai enviar o seguinte formulário para o back-end:
Nome: Kelson Email: kelson@uniesp.edu.br Curso: Sistemas para Internet
```

In the bottom right corner, there is a red logo for "uniesp" with the text "universitário" underneath it.

Vamos praticar?

- Crie um componente chamado `FormularioDeContato`
- Neste componente crie um formulário com os campos:
 - Nome
 - Contato
 - Mensagem
- Ao enviar esse formulário a função responsável pelo submit recebe esses dados em um objeto literal e converte para JSON.
 - Sintaxe: `let jsonToSend = JSON.stringify(objetoLiteral)`
- Ao fim do submit imprima através de `console.log` a mensagem: **O seguinte JSON será enviado via HTTP POST para o back-end: `${jsonToSend}`**

