Styled Components

O que é

O Styled Components é uma biblioteca popular para estilização de componentes em aplicações React. Ele oferece uma maneira intuitiva e eficiente de escrever CSS diretamente nos seus componentes React, combinando a lógica do componente com os estilos em um só lugar.

Ao invés de criar classes CSS genéricas e aplicá-las a elementos HTML, o Styled Components permite criar componentes React personalizados com estilos específicos

Documentação: https://styled-components.com/docs

Vantagens

FIAP

Maior organização e legibilidade do código

Com os estilos dentro do componente, fica mais fácil visualizar como cada parte do componente é estilizada, tornando o código mais organizado e legível.

Reutilização facilitada

Componentes estilizados podem ser facilmente reutilizados em toda a aplicação, promovendo consistência visual e facilitando a manutenção. Isso porque os estilos ficam encapsulados dentro do componente, evitando a duplicação de código e a necessidade de criar classes CSS genéricas.

Estilos dinâmicos

O Styled Components permite criar estilos dinâmicos que podem ser baseados em props ou no estado do componente. Isso oferece mais flexibilidade e permite que você crie interfaces mais responsivas e interativas.

Simplificação do gerenciamento de temas

Com o Styled Components, é mais fácil gerenciar temas de aplicação, pois você pode centralizar os estilos em um único lugar e facilmente alterá-los para diferentes temas.

Vantagens

CSS in JS

O Styled Components utiliza a abordagem CSS in JS (CSS dentro do JavaScript), que traz algumas vantagens adicionais, como:

Escopo local de estilos

Os estilos definidos com Styled Components possuem um escopo local por padrão, o que evita conflitos com outros estilos na aplicação.

Componibilidade

Você pode compor estilos de diferentes componentes para criar novos estilos, promovendo a reutilização de código.

Suporte a variáveis CSS

O Styled Components permite usar variáveis CSS, o que torna o código mais dinâmico e fácil de manter.

Usando Styled Components

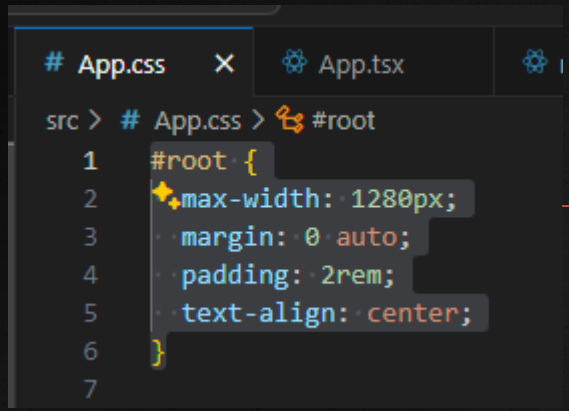
1. Vamos abrir o um projeto já criado com o Vite + React + TypeScript como já vimos em aulas anteriores. Caso não se lembre de como criar, é só executar npm create vite@latest e seguir os passos que vimos nas aulas anteriores.

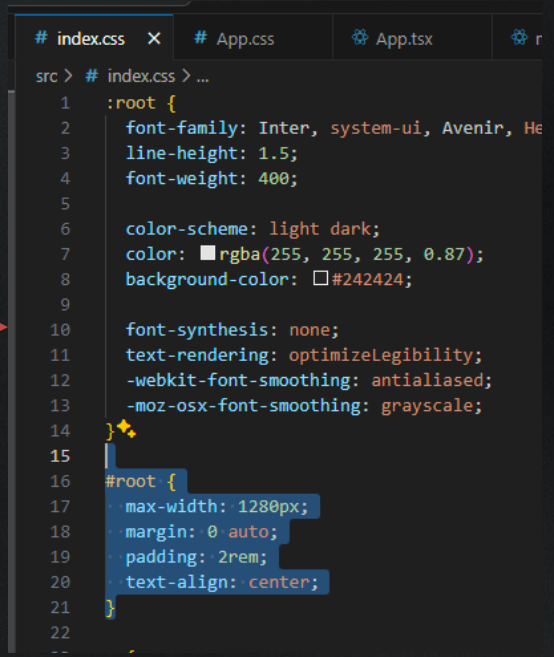
2. Agora vamos executar o comando npm install styled-components. Depois vamos iniciar nossa aplicação com o comando npm run dev e visualizar na URL localhost

```
(base) PS C:\Users\vinix\Desktop\react-first-app> npm install styled-components

added 13 packages, and audited 172 packages in 3s

41 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
(base) PS C:\Users\vinix\Desktop\react-first-app>
```

Usando Styled Components

3. Vamos mover um trecho de css que foi criado localmente pelo Vite para o escopo global. Abra o arquivo App.css e vamos recortar todo o conteúdo da regra de css que envolve o #root. Vamos mover para o index.css





Usando Styled Components

4. Agora vamos remover o import do App.css já que não vamos usar css puro com o uso do styled-components

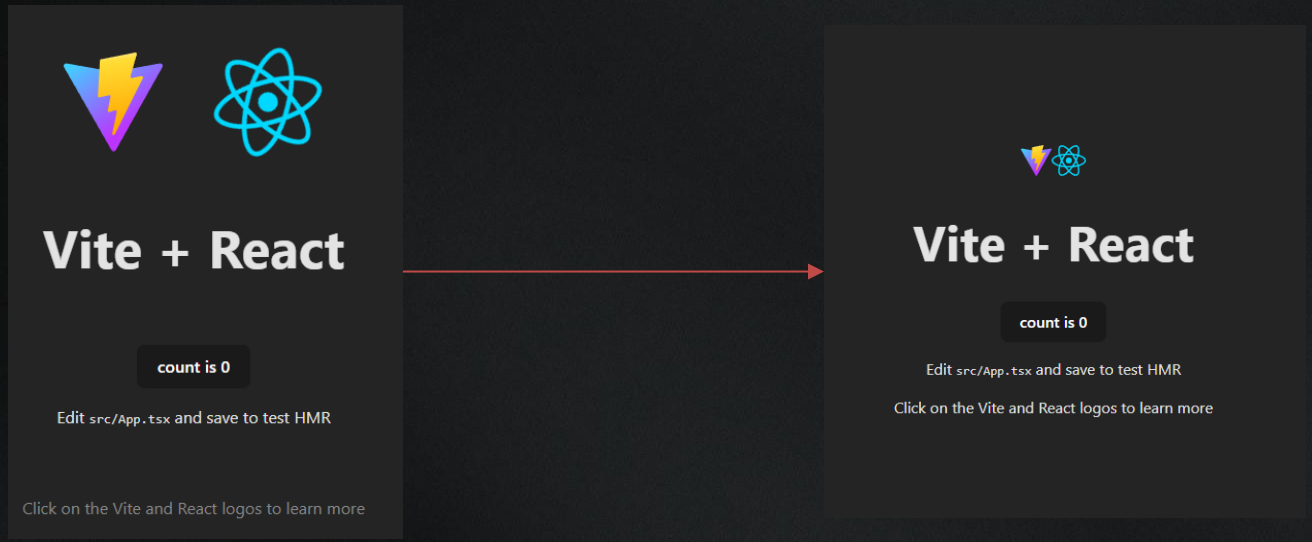
App.tsx > ...

```
1  import { useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5
```

```
1  import { useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4
5  function App() {
```

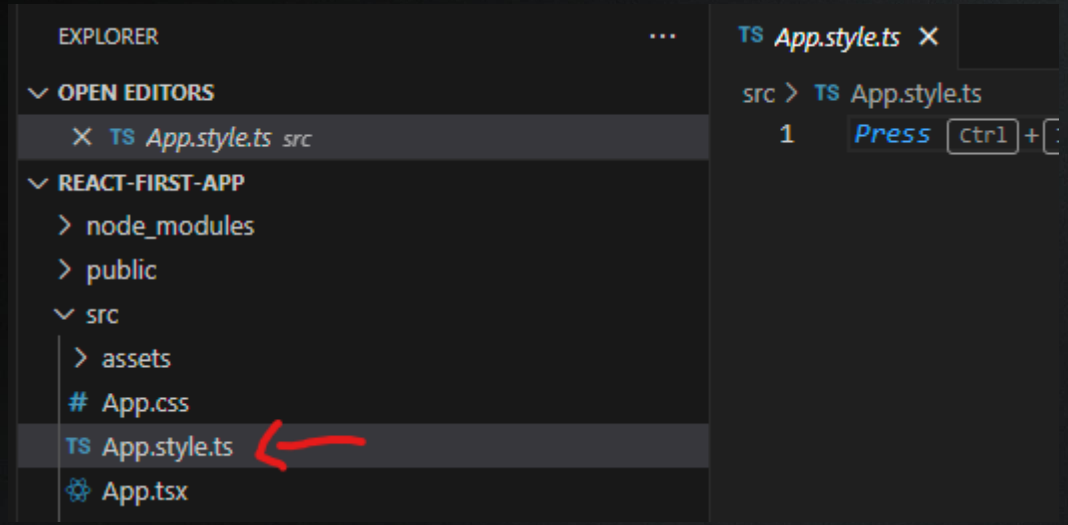
Usando Styled Components

5. Se observarmos a nossa aplicação rodando no navegador, ela mudou um pouco. Antes, o logo do React estava com uma animação e o tamanho das coisas estava diferente. Isso aconteceu porque ao remover o import anterior, removemos varias regras de CSS que estavam estilizando nossa página



Usando Styled Components

6. Agora vamos criar um arquivo App.style.tsx



Usando Styled Components

FIAP

Com o uso do Styled-components nossos estilos viram “pequenos componentes”. Nós trocaremos as tags padrão por novas tags nomeadas (que vamos nomear) com os estilos já aplicados

Usando Styled Components

7. Agora vamos transformar o nosso estilo do App.css para o formato de styled-components no App.style.tsx. A Sinaxe é simples: É como se estivéssemos exportando variaveis (const) com um estilo e uma tag especifica. Os dois código abaixo fazem a mesma coisa. Mas ainda não vai estar funcionando

src > # App.css > .logo.react:hover

```css
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}

.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}

.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }

  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}
```

App.style.tsx

src > App.style.tsx > ...

```tsx
import styled, { keyframes } from 'styled-components'

export const Card = styled.div`
  padding: 2em;
`

export const ReadTheDocs = styled.p`
  color: #888;
`

export const LogoSpinAnimate = keyframes`
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
`

export const Logo = styled.img`
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
  &:hover {
    filter: drop-shadow(0 0 2em #646cffaa);
  }
  &.react:hover {
    filter: drop-shadow(0 0 2em #61dafbaa);
  }
  &.react {
    animation: ${LogoSpinAnimate} infinite 20s linear;
  }
`
```

Usando Styled Components

FIAP

8. Depois de criado os nossos estilos precisamos usar no nosso componente. Então vamos abrir o App.tsx e importar as variaveis no jsx

src > App.tsx > ...

```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

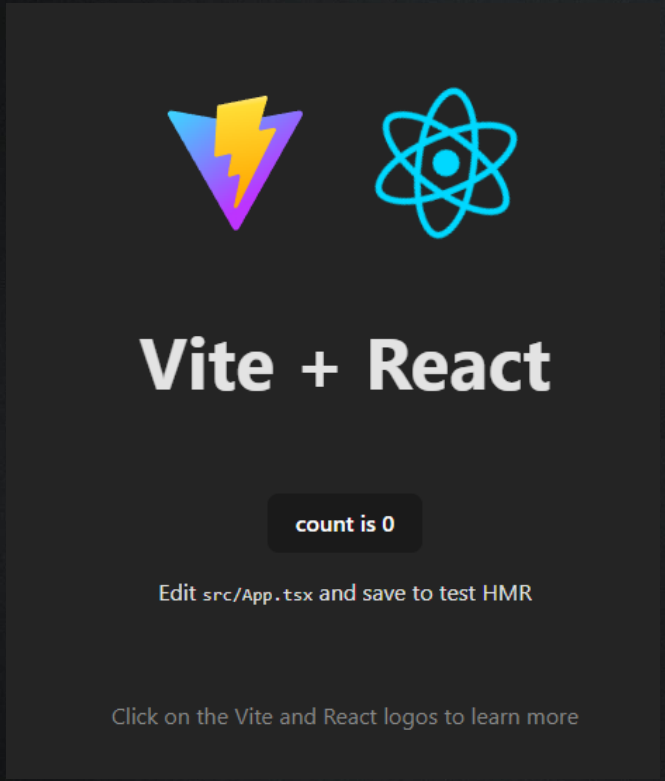
src > App.tsx > ...

```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import { Card, ReadTheDocs, Logo } from './App.style'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <Logo src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <Logo src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <Card>
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </Card>
      <ReadTheDocs>
        Click on the Vite and React logos to learn more
      </ReadTheDocs>
    </>
  )
}

export default App
```

Usando Styled Components

FIAP

Agora sim temos o resultado inicial, porém escrito com styled-components



Propriedades

Também conseguimos passar propriedades para customização dos nossos styled-components. Vamos reescrever o nosso Logo para receber uma propriedade, que vai fazer a função da classe .react que estávamos usando para fazer o logo girar e ter um hover de cor diferente. Vamos acessar a variável props do React dentro do nosso styled-component e fazer if ’s como na imagem abaixo. Como estamos utilizando TypeScript, precisamos declarar uma interface.

App.style.tsx

```
import styled, { keyframes } from 'styled-components'

export const Card = styled.div`
  padding: 2em;
`

export const ReadTheDocs = styled.p`
  color: #888;
`

export const LogoSpinAnimate = keyframes`
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
`

export const Logo = styled.img`
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
  &:hover {
    filter: drop-shadow(0 0 2em #646cffaa);
  }
  &.react:hover {
    filter: drop-shadow(0 0 2em #61dafbaa);
  }
  &.react {
    animation: ${LogoSpinAnimate} infinite 20s linear;
  }
`
```



```
import styled, { keyframes, css } from 'styled-components'

export const Card = styled.div`
  padding: 2em;
`

export const ReadTheDocs = styled.p`
  color: #888;
`

export const LogoSpinAnimate = keyframes`
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
`

export interface LogoProps extends React.ImgHTMLAttributes<HTMLImageElement> {
  reactLogo?: boolean
}

export const Logo = styled.img<LogoProps>`
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
  animation: ${props => {
    if (props.reactLogo) {
      return css`${LogoSpinAnimate} 2s linear infinite`
    }
    return "none"
  }};
  &:hover {
    filter: drop-shadow(0 0 2em ${props => props.reactLogo ? "#61dafbaa" : "#646cffaa"});
  }
`;
```

Propriedades

Agora vamos alterar o nosso HTML. Como estamos usando uma propriedade nova reactLogo para ver se é o logo do React ou do Vite, precisamos criar um Wrapper para o React não gerar nenhum erro no nosso console. Então no arquivo App.tsx vamos fazer o seguinte:

src > App.tsx > ...

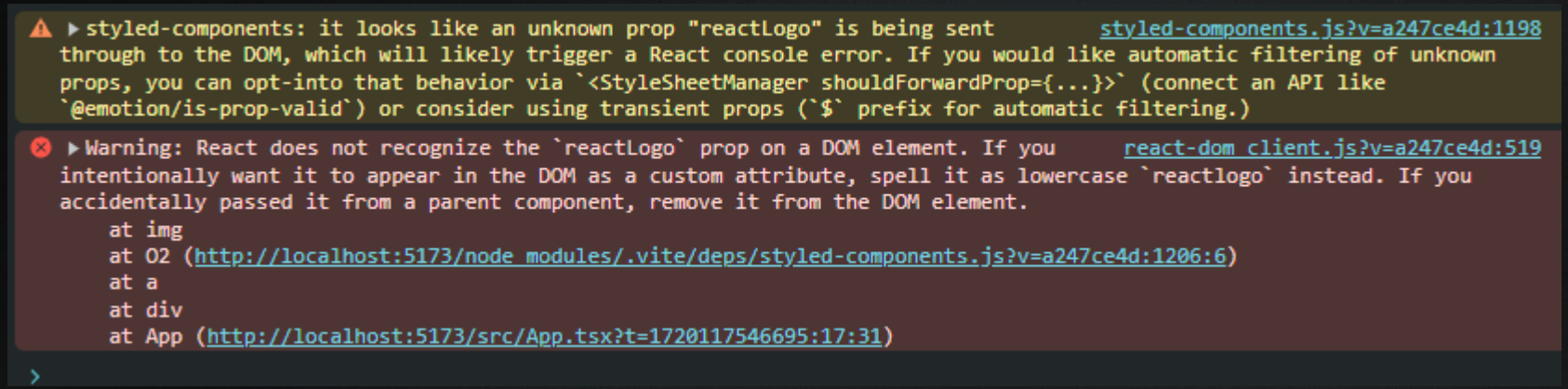
```tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import { Card, ReadTheDocs, Logo } from './App.style'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <Logo src={viteLogo} alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <Logo src={reactLogo} reactLogo={true} alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <Card>
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </Card>
      <ReadTheDocs>
        Click on the Vite and React logos to learn more
      </ReadTheDocs>
    </>
  )
}

export default App
```

src > App.tsx > ...

```tsx
import React, { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import { Card, ReadTheDocs, Logo, LogoProps } from './App.style'
import { StyleSheetManager } from 'styled-components'

const LogoWithProps: React.FC<LogoProps> = (props: LogoProps) => {
  return (
    <StyleSheetManager shouldForwardProp={(prop) => prop !== 'reactLogo'}>
      <Logo {...props} />
    </StyleSheetManager>
  )
}

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <LogoWithProps src={viteLogo} alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <LogoWithProps src={reactLogo} reactLogo alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <Card>
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </Card>
      <ReadTheDocs>
        Click on the Vite and React logos to learn more
      </ReadTheDocs>
    </>
  )
}

export default App
```

Propriedades

Com ou sem o componente StyleSheetManager o funcionamento será idêntico. Porém, sem o uso dele, vamos receber o seguinte erro no console da nossa aplicação no browser:



Então para evitar esses erros, nós criamos um sub-componente na própria página, e depois usamos ele no lugar do componente antigo, apenas para não recebermos esses erros

Dúvidas, críticas ou sugestões?


FIAP