

**Objetivos:**

- I. Styled-Components;
- II. Estilos globais com Styled-Components;
- III. Estilização de componentes existentes;
- IV. Estilização com propriedades (props);
- V. Temas.

**I. Styled-Components**

O Styled-Components é uma biblioteca JavaScript que permite a estilização de componentes React diretamente dentro do código JS ou TS, sem a necessidade de arquivos CSS separados (<https://styled-components.com/docs>).

A ideia principal por trás do Styled-Components é criar estilos encapsulados e específicos para cada componente, tornando a estilização mais fácil de gerenciar e menos propensa a conflitos. Essa abordagem é conhecida como "CSS-in-JS" (CSS dentro do JavaScript) e ganhou popularidade em projetos de desenvolvimento front-end.

Instalação do Styled-Components:

```
npm i styled-components
```

No código a seguir foi criado o componente estilizado **Button**:

- **styled.button**: cria um componente React estilizado baseado na tag `<button>`. É possível utilizar qualquer outra tag HTML, como `styled.div` ou `styled.input`;
- Estilização CSS dentro do JS: as regras CSS são escritas diretamente dentro das crases (```). A sintaxe é equivalente à utilizada em arquivos `.css`;
- Símbolo `&`: usado para referenciar o próprio elemento estilizado dentro do bloco de estilos. Ele permite adicionar pseudo-classes, pseudo-elementos ou modificadores diretamente. No exemplo, `&` representa o próprio elemento `<button>` e `:hover` é uma pseudo-classe CSS;
- Encapsulamento: os estilos aplicam-se apenas ao componente Button, garantindo que não afetem outros elementos ou componentes.

```
import styled from "styled-components";

export default function App() {
  return <Button onClick={() => alert("Olá!")}>Clique</Button>;
}

// Define o estilo do botão
const Button = styled.button`
  background-color: #4caf50;
  color: white;
  padding: 10px 20px;
  border: none;
```

```
border-radius: 5px;
cursor: pointer;

&:hover {
  background-color: #45a049;
}
;
```

Explicação do código de criação do componente estilizado:

- `const Button = styled.button`: cria um componente estilizado com base na tag `<button>`;
- Propriedades como `background-color`, `color`, `padding`, e outras são aplicadas diretamente no código para definir a estilização CSS.

## II. Estilos globais com Styled-Components

No desenvolvimento de aplicações React, há situações em que precisamos definir estilos que se aplicam de forma global a toda a aplicação, como configurações de tipografia, margens padrão, estilos de fundo, entre outros. O Styled-Components fornece o recurso `createGlobalStyle` para definir estilos globais diretamente no código JS/TS, permitindo que sejam facilmente compartilhados entre componentes.

O código a seguir faz uso de estilos globais para estilizar os elementos `body` e `p`.

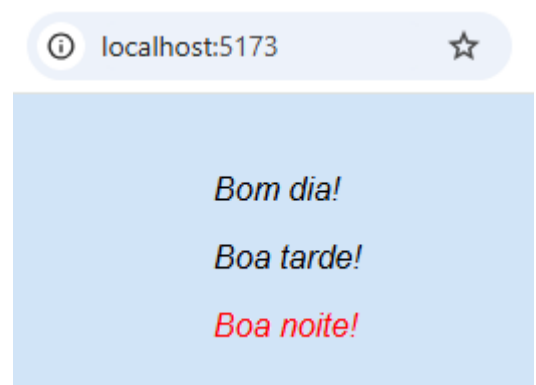
Explicação do código:

- `createGlobalStyle`: cria estilos globais que serão aplicados em toda a aplicação;
- `<GlobalStyle />`: é chamado dentro do componente App. Isso garante que as regras CSS globais sejam aplicadas assim que o componente App for montado na interface.

Diferentemente dos componentes estilizados tradicionais (como `styled.div` ou `styled.p`), os estilos criados com `createGlobalStyle` não retornam um componente visual, mas sim um conjunto de regras CSS que são injetadas globalmente no DOM assim que o componente `<GlobalStyle />` é renderizado.

Injeção no DOM:

- Quando `<GlobalStyle />` é renderizado, as regras CSS declaradas são injetadas diretamente no `<head>` do documento HTML;
- Esse processo ocorre apenas uma vez, e os estilos permanecem ativos durante todo o ciclo de vida da aplicação.



```
import styled, { createGlobalStyle } from 'styled-components';

export default function App() {
  return (
    <>
      <GlobalStyle />
      <p>Bom dia!</p>
      <p>Boa tarde!</p>
      <Texto>Boa noite!</Texto>
    </>
  );
}

const Texto = styled.p`
  color: red; /* Estilo específico para este componente */
`;

const GlobalStyle = createGlobalStyle`
  body {
    display: flex;
    flex-direction: column;
    justify-content: center; /* Alinha verticalmente */
    align-items: center; /* Alinha horizontalmente */
    height: 100vh; /* Garante que o body ocupe 100% da altura da viewport */
    margin: 0; /* Remove margens padrão do body */
    font-family: 'Arial', sans-serif;
    background-color: #d1e4f7; /* Define uma cor de fundo */
  }

  p {
    font-style: italic; /* Estilo global para todos os parágrafos */
  }
`;
```

Observações:

- `<GlobalStyle />` deve ser chamado apenas uma vez no componente raiz da aplicação (geralmente no `App`);
- Ele deve ser posicionado no início da estrutura de renderização para garantir que os estilos globais sejam aplicados corretamente.

### III. Estilização de componentes existentes

A estilização de componentes existentes com Styled-Components é um recurso que permite adicionar estilos personalizados a componentes já existentes (sejam eles provenientes de bibliotecas de terceiros ou próprios), sem a necessidade de alterar o código original do componente e preservando suas funcionalidades.

Para estilizar um componente já existente, utilizamos a função `styled()` do Styled-Components. No exemplo a seguir:

- **ColorButton**: herda todos os comportamentos e propriedades do componente **Button**;
- **BorderButton**: herda todos os comportamentos e propriedades do componente **BorderButton**.

```
import styled from "styled-components";

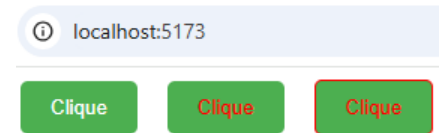
export default function App() {
  return (
    <>
      <Button onClick={() => alert("Olá!")}>Clique</Button>
      <ColorButton onClick={() => alert("Olá!")}>Clique</ColorButton>
      <BorderButton onClick={() => alert("Olá!")}>Clique</BorderButton>
    </>
  );
}

const Button = styled.button`
  background-color: #4caf50;
  color: white;
  padding: 10px 20px;
  margin: 0px 10px;
  border: none;
  border-radius: 5px;
  cursor: pointer;

  &:hover {
    background-color: #45a049;
  }
`;

const ColorButton = styled(Button)`
  color: red;
`;

const BorderButton = styled(ColorButton)`
  border: 1px solid red;
`;
```



O Styled-Components utiliza herança de estilo para adicionar novas regras ao componente, sem a necessidade de reescrever os estilos que ele já possui. Caso o componente tenha estilos internos (ou utilize estilos CSS diretos), os novos estilos podem ser aplicados de forma complementar.

No entanto, quando estilizamos um componente que já foi estilizado anteriormente, as regras de CSS definidas no Styled-Components têm precedência sobre os estilos existentes. Isso ocorre porque o Styled-Components gera

classes CSS específicas para cada instância de componente (ex.: o componente `BorderButton` sobrescreve a propriedade `border` do componente `Button`).

Vantagens da estilização de componentes existentes:

- Reutilização de componentes: permite reutilizar componentes de bibliotecas de terceiros, como Material-UI, aplicando estilos personalizados;
- Evita a duplicação de código: em vez de criar novos componentes apenas para alterar seus estilos, é possível personalizar o estilo de um único componente com Styled-Components;
- Manutenção facilitada: a estilização não afeta a estrutura ou lógica do componente original, simplificando a manutenção do código.

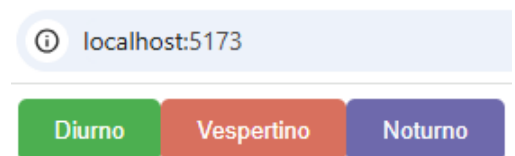
#### IV. Estilização com propriedades (props)

O Styled-Components permite alterar os estilos dinamicamente com base nas propriedades (props) recebidas. Essa abordagem proporciona maior flexibilidade e reutilização de código, permitindo que um único componente estilizado se adapte a diferentes contextos sem a necessidade de duplicar código ou criar múltiplas variações.

O código a seguir faz uso de props para definir a cor de fundo e a cor de fundo quando o cursor estiver sobre o botão.

Dentro do Styled-Components, usamos funções de interpolação para acessar as props. A função recebe as props como parâmetro e retorna o valor apropriado com base nelas (ex.: a função retornará o conteúdo da propriedade `$fundo` em (props) => props.\$fundo).

No elemento `<Button $fundo="#4caf50" $sobre="#45a049">Diurno</Button>` passamos as propriedades `$fundo` e `$sobre`, que serão usadas para personalizar o estilo do botão



```
import styled from "styled-components";

export default function App() {
  return <>
    <Button $fundo="#4caf50" $sobre="#45a049">Diurno</Button>
    <Button $fundo="#d8705e" $sobre="#dd5c45">Vespertino</Button>
    <Button $fundo="#6e69ac" $sobre="#554fad">Noturno</Button>
  </> ;
}

// Define o estilo do botão
const Button = styled.button<ButtonProps>`
  background-color: ${ (props) => props.$fundo };
  color: white;
  padding: 10px 20px;
```

```
border: none;
border-radius: 5px;
cursor: pointer;

&:hover {
  background-color: ${ (props) => props.$sobre };
}
`;

interface ButtonProps {
  $fundo: string;
  $sobre: string;
}
```

#### Observações:

- Função de interpolação: precisa estar dentro do template literal `${}` no componente estilizado. Isso permite que as props sejam avaliadas dinamicamente e o estilo seja alterado conforme necessário.
  - Em JS/TS, a interpolação é realizada usando a sintaxe `${}`, que permite inserirmos expressões, variáveis e funções dentro de uma string. No caso do Styled-Components, essa interpolação é usada para acessar propriedades (props) ou valores dinâmicos e gerar estilos CSS baseados nesses valores.
- Prefixo \$: no Styled-Components, o prefixo \$ é uma convenção recomendada para distinguir propriedades personalizadas (passadas para o componente estilizado) das propriedades padrão do HTML. Essa prática evita que as propriedades personalizadas sejam injetadas diretamente no DOM como atributos HTML inválidos. O React emite um aviso no console se uma propriedade não reconhecida for passada diretamente para o DOM;
- Tipos de Props: Props podem ser usadas com valores lógicos (booleanos) ou valores específicos (strings, números etc.). No exemplo, as props `$fundo` e `$sobre` são do tipo string, mas poderiam ser de outros tipos, dependendo da necessidade do componente.

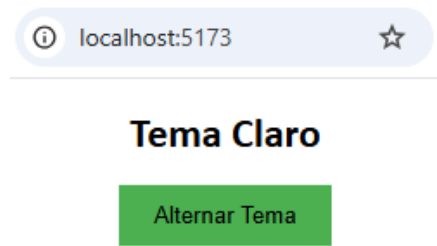
## V. Temas

O uso de temas é uma abordagem eficiente para gerenciar e padronizar estilos em uma aplicação. Com eles, é possível definir um conjunto de variáveis globais para cores, tamanhos de fonte, espaçamento e outros estilos reutilizáveis, facilitando a manutenção e garantindo consistência visual no projeto.

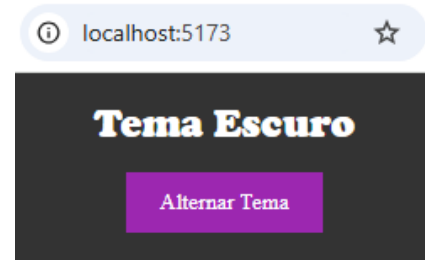
Um tema é um objeto JS que contém variáveis de estilo predefinidas. Ele permite centralizar configurações de estilo e utilizá-las em qualquer componente estilizado por meio das props do Styled-Components.

No exemplo a seguir, foram definidas as configurações de estilo para `lightTheme` e `darkTheme`, demonstrando um caso clássico de alternância entre tema claro e tema escuro.

Tema claro:



Tema escuro:



Código de exemplo:

```
import { useState } from "react";
import styled, { ThemeProvider } from "styled-components";

export default function App() {
  const [isDarkMode, setIsDarkMode] = useState(false);

  return (
    <ThemeProvider theme={isDarkMode ? darkTheme : lightTheme}>
      <Container>
        <Title>Tema {isDarkMode ? "Escuro" : "Claro"}</Title>
        <Button onClick={() => setIsDarkMode(!isDarkMode)}>
          Alternar Tema
        </Button>
      </Container>
    </ThemeProvider>
  );
}

const Container = styled.div`
  background-color: ${(props) => props.theme.colors.background};
  color: ${(props) => props.theme.colors.text};
  padding: 10px;
  text-align: center;
  min-height: 100vh;
`;

const Title = styled.h1`
  font-size: 24px;
  font-family: ${(props) => props.theme.fonts.text};
`;

const Button = styled.button`
  background-color: ${(props) => props.theme.colors.button};
  color: ${(props) => props.theme.colors.text};
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  font-family: ${(props) => props.theme.fonts.button};
`;
```

```
const lightTheme = {
  colors: {
    background: "#ffffff",
    text: "#000000",
    button: "#4caf50",
  },
  fonts :{
    text: 'Calibri',
    button: 'Arial',
  }
};

const darkTheme = {
  colors: {
    background: "#333333",
    text: "#ffffff",
    button: "#9c27b0",
  },
  fonts :{
    text: 'Cooper black',
    button: 'times new roman',
  }
};
```

Funcionamento do ThemeProvider:

- ThemeProvider: envolve os componentes da aplicação e disponibiliza o objeto de tema para todos os componentes estilizados – o **ThemeProvider** deve envolver os componentes da aplicação;
- Acesso ao tema: dentro dos componentes estilizados, as variáveis do tema são acessadas por meio de **props.theme**;
- Flexibilidade: facilita a troca de temas (ex.: modo claro e escuro) simplesmente alternando o objeto de tema.

## VI. Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercícios 1 e 2 - [https://youtu.be/f3FGiVO\\_rt4](https://youtu.be/f3FGiVO_rt4)

Exercícios 3 e 4 - <https://youtu.be/iyawSstiE9M>



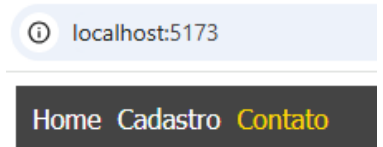
**Exercício 1** – Altere o código a seguir para que todos os estilos sejam aplicados usando Styled-Components. Adicione também a propriedade do texto ficar amarelo ao posicionar o cursor sobre o elemento.

```
export default function App() {
  return <Menu />;
}

function Menu() {
  return (
    <div
      style={{
        display: "flex",
        fontFamily: "tahoma",
        padding: "5px",
        backgroundColor: "#444",
        marginBottom: "5px",
      }}
    >
      <Item rotulo="Home" />
      <Item rotulo="Cadastro" />
      <Item rotulo="Contato" />
    </div>
  );
}

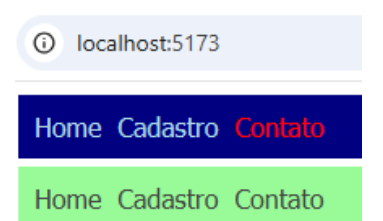
interface ItemProps {
  rotulo: string;
}

function Item({ rotulo }: ItemProps) {
  return (
    <div style={{ padding: "5px", color: "#fff", cursor: "pointer" }}>
      {rotulo}
    </div>
  );
}
```



**Exercício 2** – Altere o aplicativo do Exercício 1 para que as cores dos estilos sejam propagadas usando as propriedades definidas no elemento Menu.

```
export default function App() {
  return (
    <>
      <Menu $fundo="navy" $cor="LightBlue" $sobre="red" />
      <Menu $fundo="PaleGreen" $cor="#444" $sobre="lime" />
    </>
  );
}
```



```
);  
}
```

**Exercício 3** – Altere o aplicativo do Exercício 1 para que as cores dos estilos sejam propagadas usando tema.

**Exercício 4** – Altere o aplicativo do Exercício 3 para aplicar três temas distintos com base na largura da tela (breakpoints).

Requisitos:

- < 768px deve ser usado o tema smallTheme;
- < 1024px deve ser usado o tema mediumTheme;
- >= 1024px deve ser usado o tema largeTheme;

Dica:


- Utilizar as Media Queries diretamente no Styled Components.

```
const tema = {  
  smallTheme: {  
    fundo: "red",  
    cor: "#0ff",  
    sobre: "blue",  
  },  
  mediumTheme: {  
    fundo: "green",  
    cor: "#ff0",  
    sobre: "orange",  
  },  
  largeTheme: {  
    fundo: "blue",  
    cor: "#f0f",  
    sobre: "yellow",  
  },  
  breakpoints: {  
    small: "480px",  
    medium: "768px",  
    large: "1024px",  
  },  
};
```

Largura < 768px:



Largura >=768px e <1024px:



Largura >= 1024px:

