

Objetivos:

- I. Criar um projeto React TypeScript;
- II. Styled-components;
- III. Temas.

I. Criar um projeto React TypeScript

Siga os passos para criar uma aplicação React TS:

- a) Acesse pelo prompt do CMD o local que você deseja criar o projeto React e digite o comando a seguir para criar o projeto React usando o template para TS:

```
npx create-react-app front --template typescript
```

O projeto será criado na pasta `front`.

- b) No CMD acesse a pasta `front` e abra ela no VS Code;
- c) Ao lado tem-se a estrutura de pastas e arquivos da aplicação criada pelo CRA. Para simplificar o projeto:
 - Delete os arquivos sinalizados pela seta vermelha;
 - Substitua os códigos dos arquivos `index.html` (Figura 1), `index.tsx` (Figura 2) e `App.tsx` (Figura 3);
 - Para subir o projeto digite `npm run start` ou `npm start` no terminal do VS Code. A aplicação estará na porta padrão 3000.
- d) Adicione a dependência `npm i styled-components` (<https://www.npmjs.com/package/styled-components>);
- f) O comando `npx react-scripts start` é usado para subir o projeto React. Ele está na propriedade `scripts > start` do arquivo `package.json`. Desta forma, usamos `npm run start` ou `npm start`, no terminal do VS Code, para subir a aplicação na porta padrão 3000;

Porém, podemos definir a porta das seguintes formas:

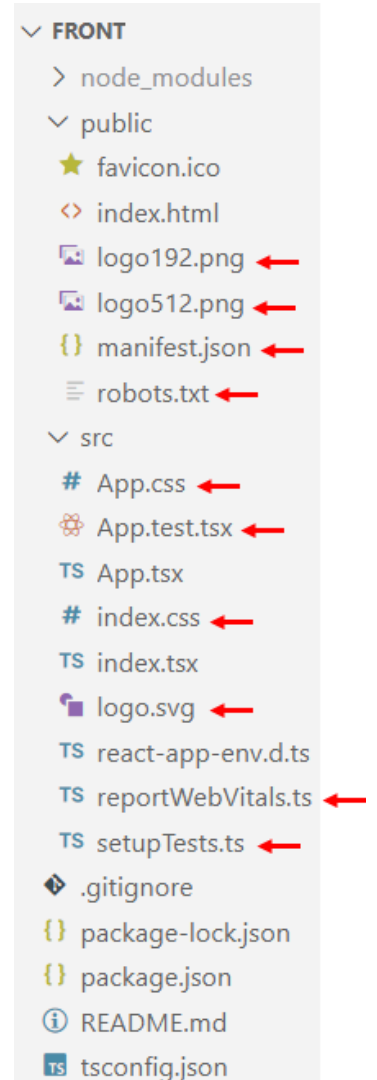
- 1) Setar o número da porta no comando `react-scripts start` da propriedade `scripts > start`, assim como é mostrado a seguir:

```
"scripts": {
  "start": "set PORT=3100 && react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

- 2) Criar o arquivo `.env` na raiz do projeto e definir a variável de ambiente `PORT` assim como é mostrado ao lado.

Como a variável `PORT` foi definida como variável de ambiente, então não precisamos colocar ela no comando `react-scripts`:

```
"scripts": {
```



```

.env
1  PORT = 3008
```

```
"start": "react-scripts start",  
"build": "react-scripts build",  
"test": "react-scripts test",  
"eject": "react-scripts eject"  
},
```

```
<!DOCTYPE html>  
<html lang="pt-br">  
  <head>  
    <meta charset="utf-8" />  
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />  
    <meta name="viewport" content="width=device-width, initial-scale=1" />  
    <title>Front</title>  
  </head>  
  <body>  
    <noscript>You need to enable JavaScript to run this app.</noscript>  
    <div id="root"></div>  
  </body>  
</html>
```

Figura 1 – Código do arquivo public/index.html.

```
import ReactDOM from 'react-dom/client';  
import App from './App';  
  
const root = ReactDOM.createRoot(  
  document.getElementById('root') as HTMLElement  
);  
root.render( <App /> );
```

Figura 2 – Código do arquivo src/index.tsx.

```
export default function App() {  
  return <div>boa noite</div>;  
}
```

Figura 3 – Código do arquivo src/App.tsx.

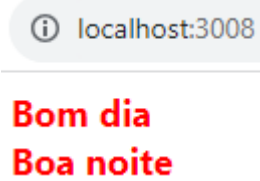
II. Styled-components

Styled-components é uma biblioteca para a estilização de componentes. Ela permite escrever CSS e anexá-lo a componentes React sem a necessidade de criar classes de estilos (<https://styled-components.com/docs>).

A principal ideia 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. Isso é conhecido como "CSS-in-JS" (CSS dentro de JavaScript), e é uma abordagem que ganhou popularidade em projetos de desenvolvimento front-end.

Tome como exemplo o componente `<Saudacao>`. Ele foi definido usando `styled.div`, que por sua vez criou o elemento `<div>` e aplicou os estilos definidos entre as aspas invertidas. Na prática será criado o seguinte elemento ao fazermos `<Saudacao>Bom dia</Saudacao>`:

```
<div style="font-family: 'Segoe UI', Verdana;color:red;font-weight:bold; font-size:18px">
  Bom dia
</div>
```

Código do arquivo src/App.tsx	Resultado no navegador
<pre>import styled from "styled-components"; export default function App() { return (<> <Saudacao>Bom dia</Saudacao> <Saudacao>Boa noite</Saudacao> </>); } const Saudacao = styled.div` font-family: 'Segoe UI', Verdana; color: red; font-weight: bold; font-size: 18px; `;</pre>	 <p>localhost:3008</p> <p>Bom dia Boa noite</p>

Dica: instale o complemento `vscode-styled-components` no VS Code. Ele ajuda no destaque de sintaxe (syntax highlighting), ou seja, ajuda na legibilidade do código, destacando elementos específicos, como palavras-chave, autocompletar e indentação dos termos. Para mais detalhes acesse <https://marketplace.visualstudio.com/items?itemName=styled-components.vscode-styled-components>.

Alguns conceitos e benefícios do styled-components:

- Estilos encapsulados: cada componente estilizado com styled-components gera classes CSS únicas para seus estilos. Isso evita vazamentos de estilos e conflitos de nome de classes;
- Sintaxe de template literal: os estilos são definidos usando uma sintaxe de template literal do JS, que permite incluir expressões JS nos estilos, tornando-os dinâmicos.
Observação: as aspas invertidas, também conhecidas como backticks (`), são um recurso do JS chamado template literals.
- Facilidade de manutenção: Como os estilos são definidos junto com os componentes, eles são fáceis de localizar e modificar. Isso melhora a manutenção do código;
- Props condicionais: podemos estilizar componentes com base em suas props. Isso é especialmente útil para criar componentes reutilizáveis e dinâmicos;
- Integração com React: podemos usar os componentes estilizados diretamente no código JSX;

- Autoprefixing: a biblioteca também lida com autoprefixing, o que significa que os estilos são automaticamente ajustados para funcionar em navegadores diferentes.

A definição de componentes estilizados utiliza a notação `styled` seguida pelo nome de um elemento HTML, por exemplo:

- `styled.button``: cria o elemento HTML `<button>`;
- `styled.h1``: cria o elemento HTML `<h1>`;
- `styled.input``: cria o elemento HTML `<input>`;
- `styled.label``: cria o elemento HTML `<label>`;
- `styled.p``: cria o elemento HTML `<p>`.

Os estilos CSS são definidos dentro do template literal (aspas invertidas), como exemplo:

```
styled.div`
  font-family: 'Segoe UI', Verdana;
  color: red;
  font-weight: bold;
  font-size: 18px;
`;
```

Componente sem corpo: o componente `<Mensagem />` não possui corpo. É uma prática comum criar uma função intermediária que encapsule o conteúdo no componente estilizado. No caso foi criada a função intermediária `Mensagem` para chamar o componente estilizado `MensagemSld`.

Código do arquivo <code>src/App.tsx</code>	Resultado no navegador
<pre>import styled from "styled-components"; export default function App() { return (<> <Mensagem /> <Mensagem /> </>); } function Mensagem(){ return <MensagemSld>Atenção!</MensagemSld> } const MensagemSld = styled.p` font-family: 'Segoe UI', Verdana; color: red; font-weight: bold; font-size: 18px; `;</pre>	<p>Atenção!</p> <p>Atenção!</p>

Componente com propriedades e sem corpo: o componente `<Pessoa nome="Ana" />` precisa ser chamado passando a propriedade `nome`. A função `Pessoa` recebe as propriedades na variável `props`. O tipo de dado da variável `props` precisa ser definido usando `interface` ou `type`. O componente estilizado `PessoaSld` não tem conhecimento da propriedade `nome`.

Código do arquivo src/App.tsx	Resultado no navegador
<pre>import styled from "styled-components"; export default function App() { return (<> <Pessoa nome="Ana" /> <Pessoa nome="Pedro" /> </>); } interface Props { nome: string; } function Pessoa(props: Props) { return <PessoaSld>Atenção! {props.nome}</PessoaSld>; } const PessoaSld = styled.p` font-family: 'Segoe UI', Verdana; color: red; font-weight: bold; font-size: 18px; `;</pre>	<p>Atenção! Ana</p> <p>Atenção! Pedro</p>

Componente com corpo: o corpo do componente `<Animal>Onça</Animal>` é recebido, por desestruturação, na variável `children` da função `Animal` e passado para o componente estilizado `AnimalSld`. O corpo pode ser texto ou marcações HTML.

Código do arquivo src/App.tsx	Resultado no navegador
<pre>import styled from "styled-components"; export default function App() { return (<> <Animal>Onça</Animal> <Animal> <p>Tatu</p> <p>Capivara</p> </Animal> </>); }</pre>	<p>Onça</p> <p>Tatu</p> <p>Capivara</p>

```

    );
  }

  function Animal({children}:any){
    return <AnimalSld>{children}</SldAnimal>;
  }

  const AnimalSld = styled.div`
    font-family: 'Segoe UI', Verdana;
    color: red;
    font-weight: bold;
    font-size: 18px;
  `;

```

Pseudoseletores: para incluir pseudoseletores na definição do estilo precisamos usar o & seguido do pseudoseletor. No exemplo a seguir o botão mudará de cor ao posicionar o cursor sobre o botão.

Observação: o caractere & é usado para se referir ao próprio componente. Isso permite estilizar o componente atual com base em seu próprio estado ou propriedades. Desta forma, quando fazemos `&:hover` estamos adicionando o estado “estar sobre”. Não podem existir espaços entre o & e :

Código do arquivo src/App.tsx	Resultado no navegador
<pre> import styled from "styled-components"; export default function App() { return (<> <ButtonSld>Salvar</ButtonSld> <ButtonSld>Limpar</ButtonSld> </>); } const ButtonSld = styled.button` background-color: green; padding: 8px; &:hover { background-color: orange; } `; </pre>	

Componente aninhado: no exemplo a seguir o componente `Header` é um wrapper (envoltório) para os elementos `label` e `input`. Observação:

- `styled.input.attrs({ type: "text" })` é usado para especificar que o elemento a ser criado tenha a propriedade `type` com valor `text`, ou seja, define um elemento `<input type="text" />`.

Código do arquivo src/App.tsx

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

export default function App() {
  return (
    <>
      <Header label="Nome" itemId="nome" />
      <Header label="Idade" itemId="idade" />
    </>
  );
}

interface Props {
  label: string;
  itemId: string;
}

function Header({ label, itemId }: Props) {
  return (
    <WrapperSld>
      <LabelSld htmlFor={itemId}>{label}</LabelSld>
      <InputSld id={itemId} />
    </WrapperSld>
  );
}

const WrapperSld = styled.div`
  margin: 5px;
  padding: 5px;
  background-color: #fff8dc;
`;

const LabelSld = styled.label`
  color: #0066b3;
  font-weight: bold;
  font-family: calibri;
  margin-bottom: 12px;
  font-size: 12px;
`;

const InputSld = styled.input.attrs({ type: "text" })`
  color: #0066b3;
  border: 1px solid #555;
  border-radius: 6px;
  padding: 8px;
  margin-left: 6px;
  font-weight: bold;
  font-family: calibri;
  font-size: 14px;
`;
```

Resultado no navegador

Nome

Idade

`;
|

Props no componente estilizado: podemos acessar as propriedades do componente dentro da template literal (delimitadas pelas aspas invertidas) da definição de estilos do componente. No exemplo a seguir a propriedade **cor** foi acessada dentro da template literal usando:

```
{props => props.cor || "#aaa"}
```

Esse código é uma arrow function que retorna o valor da propriedade **cor** (se ela existir) ou a string **"#aaa"**.

Observe que a função **Button** recebe na variável **props** as propriedades do elemento **Button** e o conteúdo **{children}**.

Código do arquivo `src/App.tsx`

Resultado no navegador

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

export default function App() {
  return (
    <>
      <Button cor="green">Salvar</Button>
      <Button>Limpar</Button>
    </>
  );
}

interface Props {
  cor?: string;
  children: React.ReactNode;
}

function Button(props: Props) {
  return <ButtonSld {...props}>{props.children}</ButtonSld>
}

const ButtonSld = styled.button<Props>`
  background-color: ${props => props.cor || "#aaa"};
  padding: 8px;

  &:hover {
    background-color: orange;
  }
`;
```

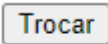
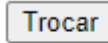


III. Temas

Os temas são uma abordagem para gerenciar variáveis e estilos que podem ser compartilhados em toda a aplicação. Eles permitem definir um conjunto de valores-chave, como cores, tipografia, espaçamento e outros estilos, que podem ser facilmente acessados e aplicados em diferentes componentes do aplicativo. O uso de temas é particularmente útil em aplicativos onde se deseja manter uma aparência consistente em toda a interface do usuário, pois os estilos são definidos em um único ponto.

O exemplo a seguir faz o uso de dois temas definidos nas variáveis claro e escuro. A seguir estão os principais passos da construção de temas usando styled-components:

- Definição dos temas: os temas são definidos usando objetos JSON. No exemplo a seguir os temas são os objetos JSON que estão nas variáveis claro e escuro. Observação: ambos os objetos JSON precisam ter as mesmas propriedades;
- ThemeProvider: o tema é propagado na árvore de componentes usando o contexto `<ThemeProvider theme={tema}>`. A propriedade `theme` recebe o objeto JSON a ser propagado pela árvore de componentes. O ThemeProvider é um componente fornecido pelo styled-components que permite envolver a árvore de componentes com um tema específico. Isso torna o tema acessível a todos os componentes filhos. No exemplo a seguir o contexto do ThemeProvider foi definido no componente App;
- Acesso às propriedades do tema: as propriedades do tema podem ser acessadas dentro da template literal (delimitadas pelas ``) da definição de estilos do componente através da propriedade theme disponível no objeto props (props.theme).

Código do arquivo src/App.tsx	Resultado no navegador
<pre>import { useState } from "react"; import styled, { ThemeProvider } from "styled-components"; export default function App() { const [tema, setTema] = useState(claro); function trocar(){ if(tema === claro){ setTema(escuro); } else{ setTema(claro); } } return (<ThemeProvider theme={tema}> <button onClick={trocar}>Trocar</button> <Titulo>Aula</Titulo> <Texto>Temas e estilos</Texto> </ThemeProvider>); } function Titulo({children}:any){ return <TituloSld>{children}</TituloSld> } function Texto({children}:any){ return <TextoSld>{children}</TextoSld> }</pre>	<p>Com tema claro:</p> <div>  </div> <p>Aula</p> <p>Temas e estilos</p> <p>Com tema escuro:</p> <div>  </div> <p>Aula</p> <p>Temas e estilos</p>

```

}

const TituloSld = styled.h3`
  font-family: Arial;
  color: ${props => props.theme.cor};
  font-style: ${props => props.theme.italico};
`;

const TextoSld = styled.p`
  font-family: cursive;
  color: ${props => props.theme.cor};
  font-style: ${props => props.theme.italico};
`;

const claro = {
  cor: "orange",
  italico: "normal"
};

const escuro = {
  cor: "#555",
  italico: "italic"
};

```

IV. Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercício 1 - <https://youtu.be/TAhRgx1i1Tg>

Exercício 2 - <https://youtu.be/wdUBg2cYa8g>

Exercício 3 - <https://youtu.be/iXnt1b10Ye8>

Exercício 1 – Usar styled-components no código a seguir para que toda a formatação CSS esteja em CSS-in-JS. O texto deverá ficar amarelo ao posicionar o cursor sobre o elemento.

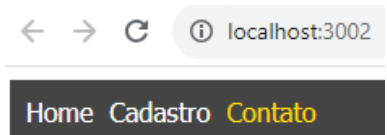
Dicas:

- Adicione a dependência styled-components;
- Use styled.div`` para criar os estilos;
- Use &:hover para criar o pseudoseletor “estar sobre”.

```

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;
}

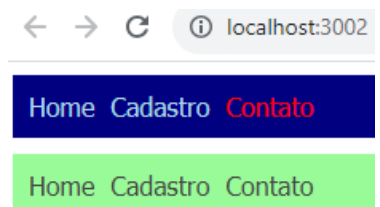
function Item({rotulo}:ItemProps){
  return (
    <div style={{padding:"5px",color:"#fff",cursor:"pointer"}}>
      {rotulo}
    </div>
  );
}
```

Exercício 2 – Alterar o Exercício 1 para que as cores dos estilos sejam propagados usando as propriedades definidas no elemento Menu.

Dicas:

- A função Menu precisa receber as propriedades fundo, cor e sobre;
- A função Item precisa receber as propriedades rotulo (corpo do item – texto Home, Cadastro ou Contato), cor e sobre;
- Dentro da template literal (aspas invertidas) acessamos as propriedades usando `${props => props.fundo}`;
- Na definição do estilo precisamos incluir as propriedades usando tipos genéricos `styled.div<Props>`.

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



Exercício 3 – Alterar o Exercício 1 para que as cores dos estilos sejam propagadas usando tema.

Dicas:

- Crie um objeto com as propriedades contendo as cores;
- Envolve a árvore de componentes usando ThemeProvider para propagar o tema pela árvore de componentes.