

ReactJS - Typescript

O que vamos ver hoje?

- O que é Typescript
- Configuração
- Tipos

Typescript o que é?

O que é Typescript?

- A Microsoft desenvolveu o TypeScript para tiparmos os dados, porém de uma forma que mantém toda a sintaxe e funcionamento do JavaScript que já conhecemos.
- No final de todo o processo, o código que desenvolvemos utilizando TypeScript irá virar JavaScript comum.

O que é Typescript?

- O TypeScript detecta quando passamos um valor diferente para uma variável declarada durante o desenvolvimento e avisa na IDE (no seu editor de código).
- Isso reflete num ambiente muito mais seguro enquanto o código está sendo digitado.

Iniciando

Iniciando

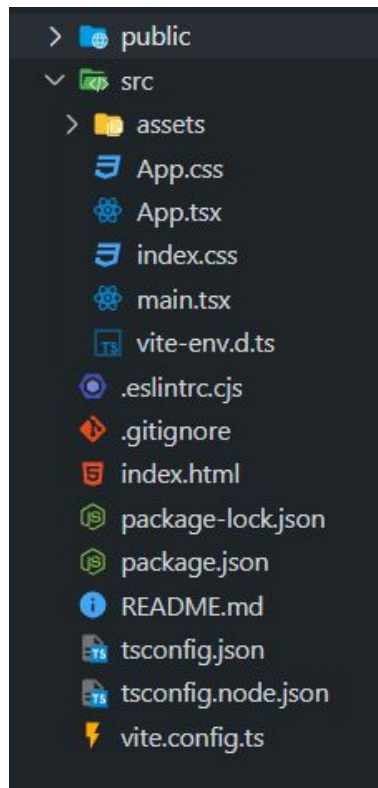
- A instalação é muito parecida com o React puro:
 - `npm init vite@latest nome-projeto --template react-ts`
 - Após, abrir o projeto no VSCode e instalar as dependências com `npm install`.
 - Importante também instalar os tipos padrões do react: `npm install --save-dev @types/react @types/node @types/react-dom`

Iniciando

- Para rodar o projeto usar o comando:
 - `npm run dev`

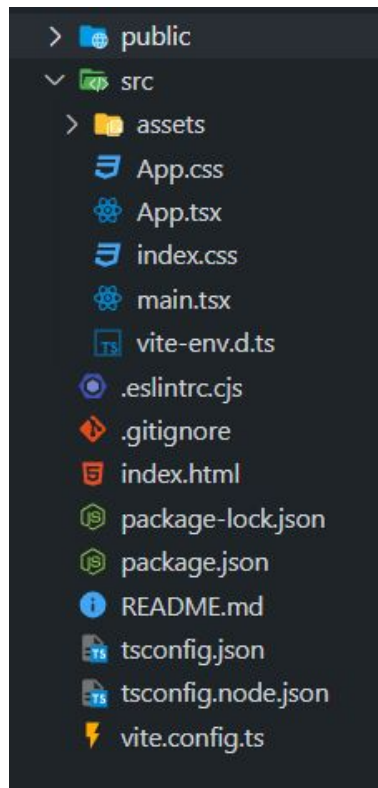
Estrutura do projeto

Estrutura do projeto



- Com Typescript a extensão dos arquivos passa a ser .tsx e .ts.
- A estrutura básica é a mesma do React com javascript.
- Há a presença de novos arquivos como:
 - tsconfig.json e tsconfig.node.json

Estrutura do projeto

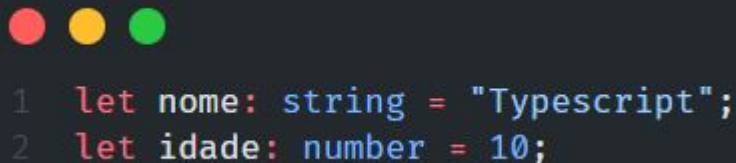


- `tsconfig.json` e `tsconfig.node.json` são os arquivos de configuração do TypeScript que definem as opções de compilação para o seu código.
- O `tsconfig.json` é usado para o código da aplicação, enquanto o `tsconfig.node.json` pode ser usado para configurar o TypeScript em ambientes Node.js;

Tipagem

Tipagem

- Basicamente ao criar uma variável devemos dizer qual o tipo dela:



```
1 let nome: string = "Typescript";  
2 let idade: number = 10;
```


Tipos

String



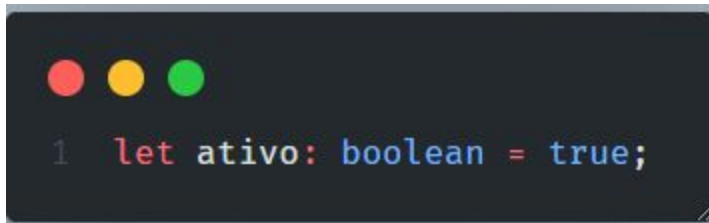
```
1 let nome: string = "Typescript";
```

Number



```
1 let idade: number = 10;
```


Boolean



```
1  let ativo: boolean = true;
```

Array



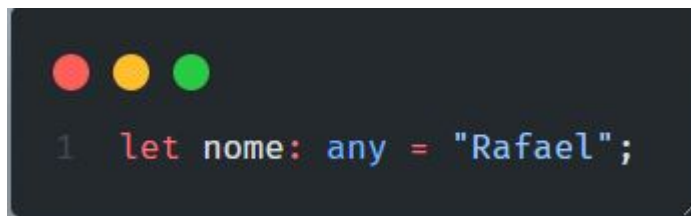
```
1 let nome: string[] = ["Rafael", "Gabriel", "Miguel"];  
2 let idades: number[] = [100, 200, 300];
```



```
1 let campos: (string | number)[] = [37, "fulano"];
```

Any

- Usado quando aceita qualquer campo ou desconhecemos o seu valor: Deve ser usada com muito cuidado!



```
1 let nome: any = "Rafael";
```

Void

- Usado para funções sem retorno:



```
1  const exibirConsole = (mensagem: string): void => {  
2      console.log(mensagem);  
3  };
```

Interfaces

- São conjuntos de métodos e propriedades que geralmente descrevem um objeto.
- A interface pode ser declarada antes da função principal ou em um arquivo separado.

```
1 interface Pessoa {  
2     nome: string;  
3     idade: number;  
4 }
```

```
1 let dev: Pessoa = {  
2     nome: "Rafael",  
3     idade: 18,  
4     };
```

Propriedades Opcionais

- Nem sempre temos certeza que um dado será exibido ou lido em nossa aplicação, para isso temos a propriedade opcional do TypeScript, que são marcadas com um "?".
- Nesses casos, os objetos da interface podem ou não definir essas propriedades.

Propriedades Opcionais



```
1 interface Pessoa {  
2   nome: string;  
3   idade: number;  
4   funcao?: string;  
5 }
```



```
1 let dev: Pessoa = {  
2   nome: "Rafael",  
3   idade: 18,  
4   };
```




```
1 let dev: Pessoa = {  
2   nome: "Rafael",  
3   idade: 18,  
4   funcao: "Programador Frontend Pleno",  
5   };
```

Extendendo Interfaces

- As interfaces podem estender uma ou mais interfaces, isso é, uma interface pode ter as propriedades declaradas em uma outra interface.
- Isso torna as interfaces e suas propriedades mais flexíveis e reutilizáveis.

Extendendo Interfaces



```
1 interface Pessoa {  
2     nome: string;  
3     idade: number;  
4 }  
5  
6 interface Desenvolvedor extends Pessoa {  
7     funcao: string;  
8     senioridade: string;  
9     tempo: number;  
10 }
```



```
1 let dev: Desenvolvedor = {  
2     nome: "Rafael",  
3     idade: 18,  
4     funcao: "Programador",  
5     senioridade: "Pleno",  
6     tempo: 5,  
7 };
```

Eventos



```
1  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {  
2    console.log(e.target.value);  
3  };  
4  
5  const handleSubmit = (e: React.SyntheticEvent) => {  
6    e.preventDefault();  
7  };
```

Props

O tipo '{}' não tem as propriedades a seguir do tipo 'Props': nome, funcao ts(2739)

```
(alias) const MaterialCard: FC<Props>  
import MaterialCard
```

Props



```
1  import "../App.css";
2  import MaterialCard from "../components";
3
4  function App() {
5      return <MaterialCard nome="Rafael" funcao="Programador" />;
6  }
7
8  export default App;
9
```

Props

```
1 import Card from "@mui/material/Card";
2 import CardContent from "@mui/material/CardContent";
3 import Typography from "@mui/material/Typography";
4 import { FC } from "react";
5
6 interface Props {
7   nome: string;
8   funcao: string;
9 }
10
11 const MaterialCard: FC<Props> = ({ nome, funcao }) => {
12   return (
13     <Card sx={{ minWidth: 275 }}>
14       <CardContent>
15         <Typography>{nome}</Typography>
16         <Typography>{funcao}</Typography>
17       </CardContent>
18     </Card>
19   );
20 };
21
22 export default MaterialCard;
```

Resumo

Resumo



- React com Typescript deixa o processo de codificação mais consistente evitando erros na atribuição de variáveis.

Dúvidas?



Programa
3000 TALENTOS TI
Obrigado(a)!