
CURSO DE TYPESCRIPT - PORTFÓLIO

DEV

PROFESSORES



Crystian Printes

Software Developer | Tech Lead

Projeto: QAP

Linkedin : www.linkedin.com/in/crystian-printes-b052691b7

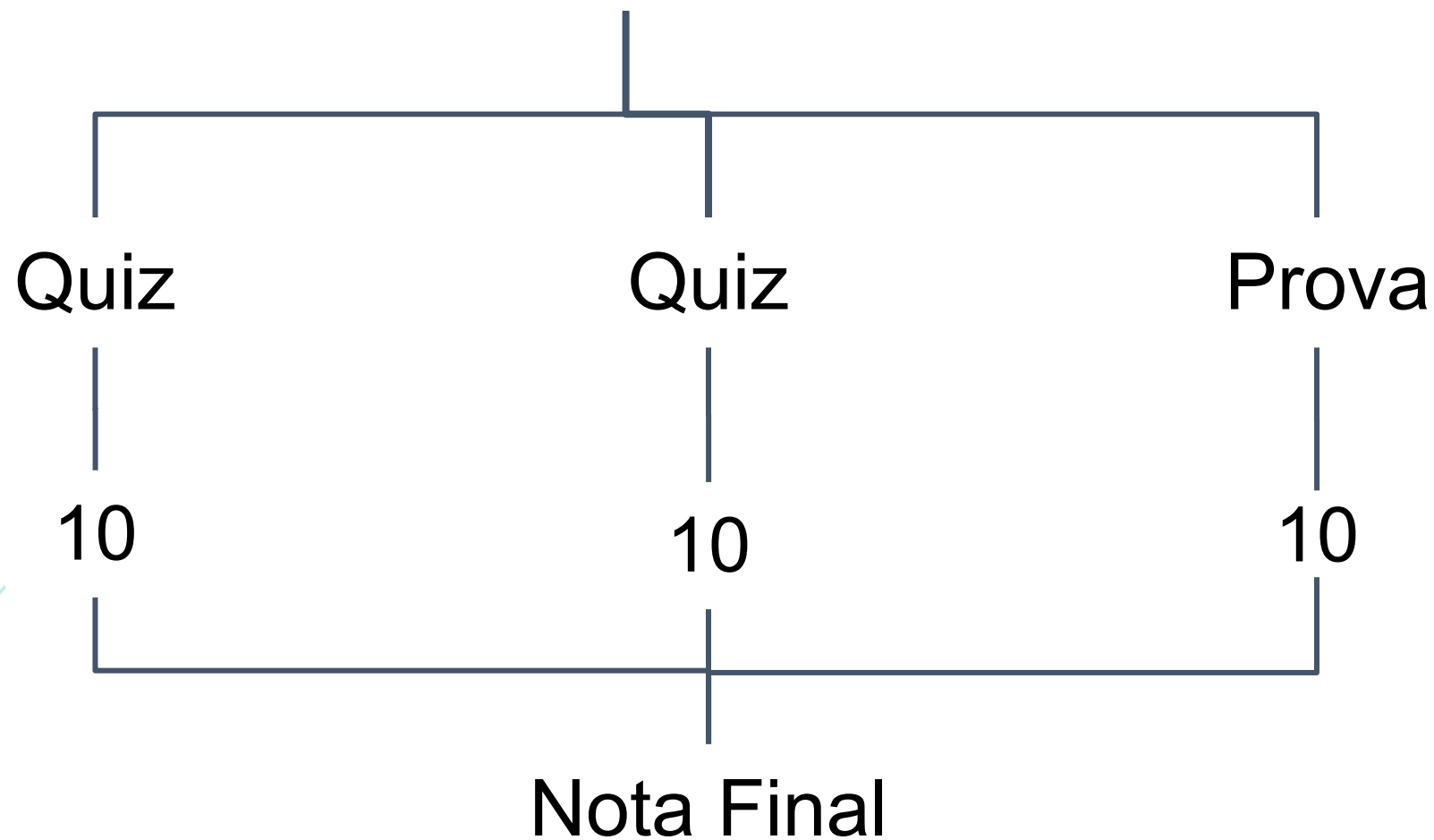
Wagner Sampaio

Desenvolvedor Front-end

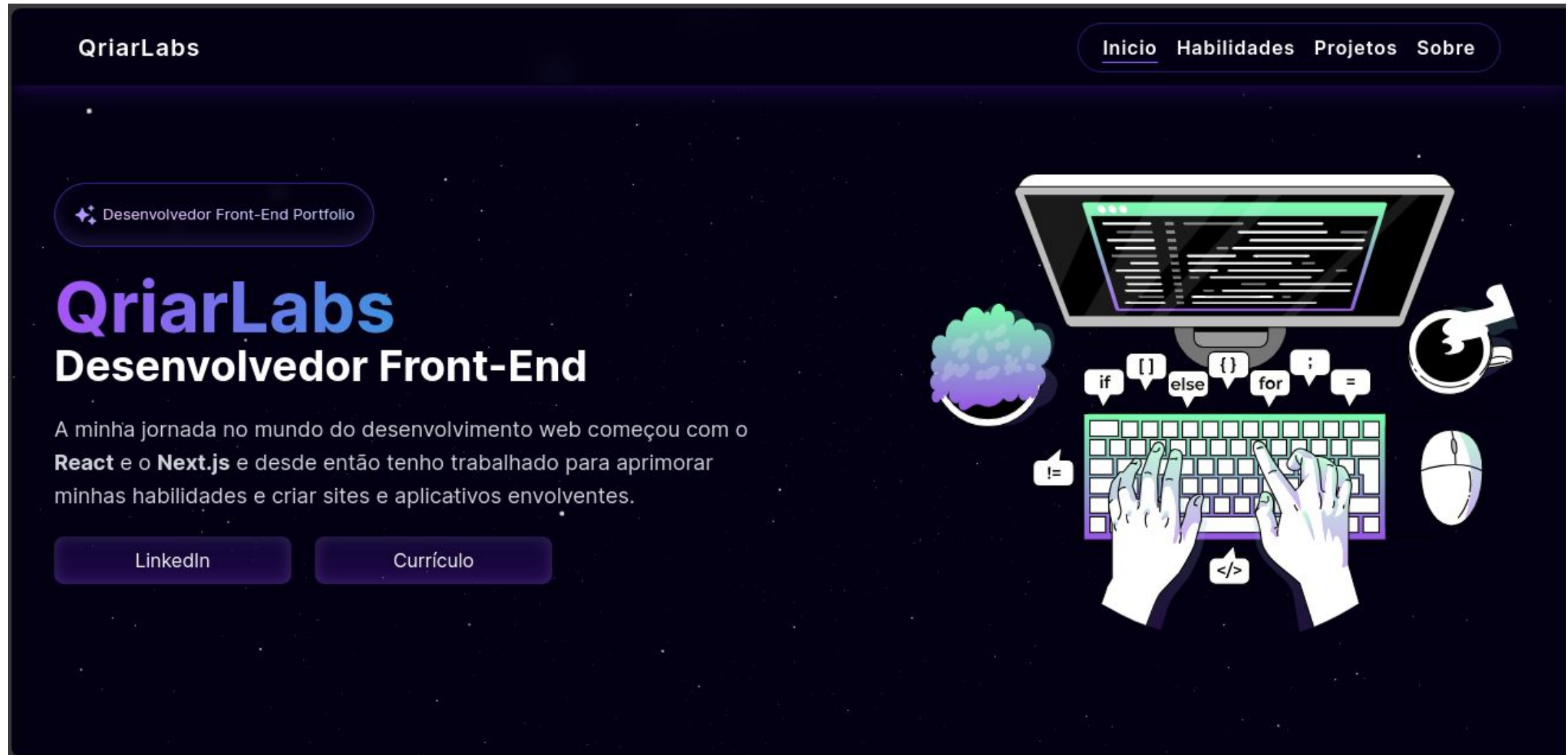
Projeto: Qscim | QA

Linkedin: <https://www.linkedin.com/in/wagsampaioner-viana--bb80ab207/>

AVALIAÇÃO



PROJETO - PORTFOLIO DEV



INTRODUÇÃO HTML CSS e JS

- **HTML** (Hypertext Markup Language) é uma linguagem de marcação usada para criar e estruturar páginas web. Ela consiste em uma série de elementos ou "tags" que envolvem o conteúdo para definir sua função e aparência.



INTRODUÇÃO HTML CSS e JS

- **CSS** (Cascading Style Sheets) é uma linguagem utilizada para estilizar e dar estilo a elementos HTML em uma página web. Com o CSS, os desenvolvedores podem controlar aspectos visuais como cor, fonte, espaçamento e layout das diferentes partes de uma página, criando uma experiência de usuário mais atraente e consistente



INTRODUÇÃO HTML CSS e JS

- **JavaScript** (JS) é uma linguagem de programação utilizada principalmente para criar interatividade em páginas web. Com o JavaScript, os desenvolvedores podem manipular elementos HTML, responder a eventos do usuário, criar animações, validar formulários e muito mais.



TYPESCRIPT: O QUE ELE RESOLVE?

- TypeScript é essencialmente um superset de JavaScript, focado em elevar o nível da linguagem. Ele introduz não apenas tipagem estática, mas também traz o paradigma de Orientação a Objetos, simplificando e tornando viável o desenvolvimento de software robusto e de longa duração, algo que o JavaScript não suportava anteriormente.




TypeScript

vs



JavaScript

TYPESCRIPT: O QUE ELE RESOLVE?



```
1  function soma(a, b) {  
2      return a + b  
3  }  
4  
5  console.log( soma(1, 1)) //2  
6  console.log( soma('1', '1')) //11
```


TYPESCRIPT: O QUE ELE RESOLVE?

```
src/1-tipos-basicos/index.ts:6:18 - error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.
```

```
6 console.log(soma('1', '1'))
```



TYPESCRIPT: O QUE ELE RESOLVE?



```
1 function soma(a: number, b: number) {  
2     return a + b  
3 }  
4  
5 console.log( soma(1, 1)) //2  
6 console.log( soma('1', '1')) //11
```



VARIÁVEIS


Para declarar variáveis no TypeScript podemos usar de três formas, `var`, `let` e `const`.

- `var : type = Valor;`
- `let : type = Valor;`
- `const : type = Valor;`

Podemos inferir ou não valores às variáveis durante sua declaração.

VARIÁVEIS

- Podem receber funções como valores:



```
1  let x = function () {  
2      return "Ferias!!!";  
3  }  
4  console.log(x());  
5  
6  //Ferias!!!
```

TIPOS PRIMITIVOS


- **Number:** Representa valores de ponto flutuante IEEE 754 de precisão dupla de 64 bits.
- **Boolean:** Valores true e false.
- **String:** Sequência de caracteres armazenados como unidades de código Unicode UTF-16
- **Any:** Assumem qualquer valor de tipos possíveis e desativa a checagem estática.

FUNÇÕES

- Funções nomeadas
- Funções anônimas
- Arrow functions

TS Functions

FUNÇÃO - NOMEADA



```
1  function MinhaFuncao(x: number, y: number)
2      return (x * y);
3  }
4
5  console.log(MinhaFuncao(3,5))
```


FUNÇÃO - ANÔNIMA



```
1  var anonimo = function () {  
2      console.log("Aula TS");  
3  }
```

ARROW FUNCTION



```
1 // Soma dois números usando uma arrow function
2 const sum = (x: number, y: number): number => x + y;
3
4 // Exemplo de uso:
5 const result = sum(10, 20); // Retorna 30
6 console.log(result); // Exibe 30 no console
```

TIPAGENS EM FUNÇÃO

```
1  interface valores {  
2      x: number;  
3      y: number;  
4  }  
5  
6  const somarArrow = (valor: valores) => {  
7      return(  
8          console.log(valor.x + valor.y)  
9      )  
10  
11  }  
12  
13  const valores: valores = { x: 5, y: 3 };  
14  somarArrow(valores);
```

AWAIT - ASYNC

- **Async/await** é uma maneira de lidar com operações assíncronas de forma mais síncrona e legível. Ele permite que você escreva código que parece sequencial, mas ainda lida com tarefas assíncronas nos bastidores.
- **Promessas** são a base para async/await. Uma promessa representa um valor que pode estar disponível agora, no futuro ou nunca. Ela é usada para tratar operações assíncronas.

AWAIT - ASYNC

```
1 // Função assíncrona que simula uma busca na web
2 async function fetchUserData(userId: number): Promise<string> {
3   try {
4     const response = await fetch(`https://api.example.com/users/${userId}`);
5     const data = await response.json();
6     return data.name;
7   } catch (error) {
8     console.error('Erro ao buscar dados do usuário:', error);
9     return 'Nome não encontrado';
10  }
11 }
12
13 // Exemplo de uso
14 const userId = 123;
15 fetchUserData(userId)
16   .then((name) => {
17     console.log(`Nome do usuário com ID ${userId}: ${name}`);
18   })
19   .catch((error) => {
20     console.error('Erro geral:', error);
21   });
```

AWAIT - ASYNC

- **fetchUserData** é uma função assíncrona que busca dados de um usuário pelo ID.
- Usamos **await** para esperar que a resposta da API seja processada.
- Se ocorrer algum erro, capturamos e tratamos com **try/catch**.
- O resultado é uma **promessa** que resolve com o nome do usuário ou uma mensagem de erro.

BORA LÁ? - PREPARANDO O AMBIENTE

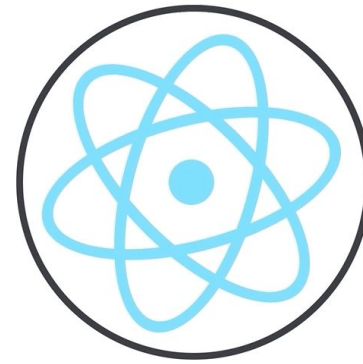
O QUE VAMOS USAR?



NEXTJS



TAILWIND



REACT



NODE

NÃO ERA SÓ TYPESCRIPT?



INCIANDO O PROJETO



```
1 npx create-next-app@latest my-project --typescript --eslint
2 cd my-project
```



```
1 npx create-next-app@latest my-project --typescript --eslint
2 cd my-project
```

LINKS

- * Playground typescript: [Clique aqui](#)
- * Tailwind instalação com NEXTjs: [Clique aqui](#)