**Sobre o compilador do TypeScript**

O uso do Node.js não é opcional, pois o compilador depende desta plataforma para funcionar, inclusive é baixado pelo seu gerenciador de pacotes npm. Além disso, o código TypeScript precisa ser traduzido/convertido para um código em ECMASCRIPT para que seja entendido pelo navegador. Lembre-se que apenas ECMASCRIPT é suportado pelo navegador.

O Node.js é usado apenas em ambiente de desenvolvimento, para que possamos gerar os arquivos .js a partir da compilação dos arquivo .ts.

|  |
| --- |
| Inicial   1. No terminal na pasta do projeto dê o comando npm init → tecle várias vezes Enter até ele terminar: Este comando é para ele criar automaticamente o arquivo package.json que é um arquivo onde todas as dependências do npm ficaram listadas 2. No terminal na pasta do projeto dê o comando npm install typescript@2.3.2 --save-dev 3. Criar o arquivo tsconfig.json com as configurações mínimas, este arquivo informa em qual versão do ecma queremos converter os arquivos TypeSrcipt 4. No arquivo package.json incluir no script o comando para ao rodar, chamar o compilador do typeScript 5. Para rodar o compilador usar o comando npm run compile |

**Sobre casting explícito**

Em TypeScript, podemos referenciar um dado de um tipo mais especializado através de um tipo mais genérico. Por exemplo:

let x: Element;

let y: HTMLInputElement;

x = y; // funciona!

O código acima é possível, porque todo HTMLInputElement é um Element. O que ocorre é um casting implícito, no qual o desenvolvedor não precisa atuar. Contudo, nem todo Element é um HTMLInputElement.

Por isso não podemos fazer:

let x: Element;

let y: HTMLInputElement;

y = x; // erro

Temos o erro:

Type 'Element' is not assignable to type 'HTMLInputElement'.

Property 'accept' is missing in type 'Element'.

Contudo, quando usarmos document.querySelector() o TypeScript considera que o retorno será sempre do tipo Element. Justo, pois document.querySelector pode retornar um HTMLInputElement, HTMLTableElement, HTMLAnchorElement, etc. O que todos eles possuem em comum, são elementos do DOM, por isso podem ser tratados como Element.

Contudo, Element não expõe propriedades e métodos de cada um dos tipos específicos que listamos no parágrafo anterior e isso pode nos causar problemas em nosso código, por exemplo, para pegar o value de um HTMLInputElement.

Aprendemos que podemos realizar um casting **explícito** de um tipo mais genérico para um tipo mais específico.

let tabela = <HTMLTableElement> document.querySelector('table');

Realizamos o casting explícito de Element para HTMLTableElement. Inclusive, devido ao casting, o TypeScript infere que o tipo de tabela será HTMLTableElement.