**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.

**Instalando TypeScript Definitions**

No caso, vamos instalar o tipo do jQuery. Vale lembrar que esse tipo não foi definido pela equipe do jQuery:

npm install @types/jquery@2.0.42 --save-dev

Você pode instalar o type definiton da sua biblioteca favorita, contanto que ela exista. Inclusive podem haver arquivos de definições criados por mais de um colaborador, no final, somos nós que devemos escolher o arquivo que for mais atualizado. Não há solução mágica, é necessário realizar pesquisas na Internet. Inclusive, pode ser que nem existe um arquivo tsd para sua biblioteca favorita, sendo assim, a solução com declare var continua sendo válida.

Vamos abrir e fechar o Visual Studio Code. Por padrão, o compilador do TypeScript procurará por padrão todos os @types dentro da pasta node\_modules sem termos que nos preocupar.

**Para saber mais: o repositório @types**

No npm, existe uma série de TypeScript definitons files para as mais diversas bibliotecas e frameworks do mercado. Por exemplo, se quisermos instalar o tsd do jQuery, acessamos

<https://www.npmjs.com/package/@types/jquery>

Se quisermos do lodash ou underscore acessamos

https://www.npmjs.com/package/@types/lodash

https://www.npmjs.com/package/@types/underscore

Dessa forma, antes de sair buscando pela internet os arquivos tsd que precisamos, podemos tentar a sorte executando o comando:

npm install @types/nomeDaLibOuFramework --save-dev

Nesse sentido, se quisermos instalar os tsd das três bibliotecas que foram citadas, fazemos:

npm install @types/jquery --save-dev

npm install @types/lodash --save-dev

npm install @types/underscore --save-dev

Qualquer tsd files que esteja dentro de node\_modules/@types será lidado automaticamente pelo compilador do TypeScript.

É preciso se conformar quando não houver do Typing para sua biblioteca preferida, neste caso, a estratégia do declare var que vimos neste treinamento é uma saída, não muito ideal, mas que permitirá seu código compilar até que você encontre seu tsd.

**Extirpando os comentários do processo de compilação**

Podemos resolver esse problema facilmente solicitando ao compilador do TypeScript que remova todos os comentários do arquivo JavaScript resultante da compilação do nosso código TypeScript. Para isso, basta adicionar a propriedade "removeComments": true no arquivo tsconfig.json.