TypeScript – Uma compilação melhor – Sem comentários

Não vale a pena manter comentários do código fonte **TypeScript** no código compilado para javaScript, então dá pra remover eles adicionando uma **configuração no *"tsconfig.json"***:

```
tsconfig.json > ...
{
    "compilerOptions": {
        "outDir": "dist/js",
        "target": "ES6",
        "noEmitOnError": true,
        "noImplicitAny": true,
        "removeComments": true,
    },
    "include": ["app/**/*"]
}
```

TypeScript – Uma compilação melhor – Strict Null Checks

Outra **configuração bem interessante é a "stricNullChecks"**, vamos supor que um **determinado valor pode ter tipos diferentes**, por exemplo, quando usamos o ***querySelector*** sabemos que vamos ter um **HTMLElement, mas se o parâmetro que eu passar for errado eu vou ter um null.**

Casting

Quando eu habilito a configuração de "***strictNullChecks***" eu estou dizendo pro **TS** me forçar a tratar esses possíveis **nullos:**

```
tsconfig.json > ...
{
    "compilerOptions": {
        "outDir": "dist/js",
        "target": "ES6",
        "noEmitOnError": true,
        "noImplicitAny": true,
        "removeComments": true,
        "strictNullChecks": true
    },
    "include": ["app/**/*"]
}
```

```
constructor(){
    this.inputData = document.querySelector('#data');
    this.inputQuantidade = document.querySelector('#quantidade');
    this.inputValor = document.querySelector('#valor');
    this.negociacoesView.atualizar(this.negociacoes);
}
```

Uma **forma de tratar esses possíveis nullos** é fazendo **casting**, isso já existe em outras linguagens, mas significa **basicamente forçar que o valor passado vai ter o tipo determinado do atributo**, é como se falássemos *"eu to garantindo que o tipo recebido vai poder ser convertido para esse valor aqui"*:

```
export class NegociacaoController{

    private inputData : HTMLDataElement;
    private inputQuantidade : HTMLInputElement;
    private inputValor : HTMLInputElement;

    constructor(){
        this.inputData = document.querySelector('#data') as HTMLDataElement;
        this.inputQuantidade = document.querySelector('#quantidade') as HTMLInputElement;
        this.inputValor = document.querySelector('#valor') as HTMLInputElement;

        this.negociacoesView.atualizar(this.negociacoes);
    }
```

Tratar o possível null

Outra abordagem diferente do casting é tratar o possível null, se **colocarmos o elemento que pode ser null** dentro de um **"if"** o **TypeScript** consegue entender que esse cara **vai cair em determinado código apenas se ele não for null:**

```
const form = document.querySelector('.form');

if(form){
    form.addEventListener('submit', event => {
        event.preventDefault();
        controller.adiciona();
    })

} else {

    throw Error('Elemento não encontrado')
}
```