09

Para saber mais: TypeScript e Node.js

TypeScript não é uma linguagem exclusiva para frontend, ela pode ser usada também no backend com Node.js. Contudo, como existem milhares (sem exagero) de módulos criados no repositório do npm (um dos maiores do mundo), as chances dos módulos da sua aplicação não terem seu respectivo TypeScript Definition file são gigantes. A única garantia que você terá são as definições dos módulos padrões do Node.js:

```
npm install @types/node --save-dev
```

Como fica o tsconfig.json?

Outro ponto importante, aliás, uma dica, é evitarmos o uso do `strictNullChecks` e do `noImplicityAny`. Caso estejam presentes no arquivo `tsconfig.js` seus valores devem ser `false`. A ativação dessas configurações poderá gerar inúmeros problemas com possíveis definições que você venha a baixar.

Como fica o sistema de módulos?

Os módulos do Node.js usam o padrão commonjs. Felizmente o compilador TypeScript aceita este parâmetro na propriedade `module` do arquivo `tsconfig.json`.

Vejamos um exemplo de arquivo que configura o TypeScript para um ambiente `Node.js`:

```
{
    "compilerOptions": {
        "target": "es6",
        "outDir": "js",
        "noEmitOnError": true,
        "noImplicitAny": false,
        "removeComments": true,
        "module": "commonjs",
        "strictNullChecks": false,
        "experimentalDecorators": true
    },
    "include": [
        "ts/**/*"
    ]
}
```

Há mais um detalhe importante.

Como realizaremos os imports?

Vejamos um código escrito sem TypeScript. Ele nada mais faz do que criar um novo arquivo no sistema de arquivos:

```
const { writeFile } = require('fs');
```

```
writeFile('teste.txt', 'Gravei no arquivo', err => {
    if(err) {
        return console.log('Não foi possível criar o arquivo');
    }
    console.log('arquivo criado com sucesso');
});
```

No entanto, quando estamos usando TypeScript, precisamos mudar a forma com que importamos nossos módulos:

```
// veja a diferença

import { writeFile } from 'fs';

writeFile('teste.txt', 'Gravei no arquivo', err => {
    if(err) {
        return console.log('Não foi possível criar o arquivo');
    }
    console.log('arquivo criado com sucesso');
});
```

Por debaixo dos panos o TypeScript transcompilará o arquivo para:

```
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
const fs_1 = require("fs");
fs_1.writeFile('teste.txt', 'Gravei no arquivo', err => {
    if (err) {
        return console.log('Não foi possível criar o arquivo');
    }
    console.log('arquivo criado com sucesso');
});
```

Para um codebase já existente, realizar essa mudança pode ser algo muito custoso.

Exemplo para download

Você pode baixar o exemplo apresentado neste (https://s3.amazonaws.com/caelum-online-public/typescript/ts_node.zip) link. Para testá-lo, descompacte-o e execute dentro da pasta `ts_node` o comando `npm install`. Depois, `npm run compile` para compilar o arquivo. Por fim, entre dentro de `ts_node/js` e execute o arquivo `local.js` com a instrução `node local`.