**Transcrição**

Para que possamos utilizar o TypeScript precisamos da plataforma Node.js instalada. Aliás, essa plataforma já era um requisito de infraestrutura apontado no exercício obrigatório do capítulo.

É através do gerenciador de pacotes do Node.js que instalamos o TypeScript, mas primeiro, precisamos criar o arquivo package.json que nada mais é do que uma "caderneta" na qual temos registrados todos os módulos da aplicação baixados pelo npm.

Através do seu terminal favorito, vamos acessar a pasta alurabank. Dentro dela, vamos executar o comando:

npm init

Podemos teclar ENTER para todas as perguntas. No final, teremos o arquivo alurabank/package.json:

{

"name": "alurabank",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

Agora que temos nosso arquivo criado, vamos solicitar ao npm que instale o TypeScript para nós:

npm install typescript@2.3.2 --save-dev

Dentro de instantes ele será instalado dentro da pasta alurabank/node\_modules. Esse passo não é suficiente, precisamos configurar o compilador. Aliás, muitas IDE's escondem esses detalhes do desenvolvedor, mas inevitavelmente cedo ou tarde ele terá que lidar com essas configurações para poder alterar o comportamento do compilador do TypeScript. Este curso, mesmo sendo introdutório, o deixará seguro com tudo o que acontece por debaixo dos panos.

O próximo passo será renomearmos a pasta alurabank/app/js para alurabank/app/ts, inclusive vamos mudar a extensão dos arquivos app.js e Negociacao.js respectivamente para app.ts e Negociacao.ts. Afinal, a extensão .ts é aquela de todo arquivo TypeScript.

**O arquivo tsconfig.json**

Precisamos criar o arquivo alurabank/tsconfig.json que guardará as configurações do nosso compilador.

{

"compilerOptions": {

"target": "es6",

"outDir": "app/js"

},

"include": [

"app/ts/\*\*/\*"

]

}

Nele, indicamos em compilerOptions as configurações do compilador. No caso, indicamos que o resultado final da compilação será um código compatível com es6 e que eles ficarão dentro da pasta app/js. Por fim, em include, indicamos o local onde o compilador deve buscar seus arquivos.

Excelente, temos a configuração mínima para que nosso compilador funcione, mas como o executaremos? Uma boa prática é criarmos um script em nosso package.json que se encarregará de chamá-lo para nós através do terminal.

Vamos alterar alurabank/package.json e adicionarmos o script:

"compile": "tsc"

Nosso package.json ficará assim:

{

"name": "alurabank",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"compile": "tsc"

},

"author": "",

"license": "ISC",

"devDependencies": {

"typescript": "^2.3.2"

}

}

Feche e abra o VSCode para que ele possa levar em consideração as configurações que realizamos no compilador.

Agora, através do terminal, ainda dentro da pasta alurabank faremos:

npm run compile

Após a conclusão do comando, veremos uma série de mensagens de erro, inclusive no próprio visual studio code veremos as mesmas mensagens em todo lugar que estiver sublinhado de vermelho. Isso significa que houve algum problema de compilação do nosso código que precisamos resolver, ou seja, alguma sintaxe não compatível com TypeScript.

**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/loadash --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.