07

Instalação e configuração do compilador

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.