07

Organizando nosso código em módulos

Transcrição

Vamos começar com o uso de uma boa prática que inclusive é preconizada e indicada no site do Angular. Atualmente, temos um único componente, o qual faz parte do módulo `AppModule`, também denominado ***root module*** ("módulo raiz") no Angular, por ser o primeiro a ser carregado pela aplicação.

Ao longo do nosso projeto, criaremos melhorias e outros componentes que dizem respeito ao universo de imagens. E já que um componente não funciona se não pertencer a um módulo, teremos que inclui-los em `declarations` de `app.module.ts` e, eventualmente, pelo seu tamanho excessivo, acabaremos não conseguindo distinguir estes grupos com tanta clareza.

Sendo assim, existe a possibilidade de criarmos um ***feature module***, um módulo que contém vários componentes que fazem sentido de serem agrupados. Então, em "app", criaremos a pasta "photos", e tudo que se relaciona com as imagens ficarão contidos nela. Inclusive, incluiremos dentro dela a pasta "photo", e automaticamente — já que o Angular utiliza o TypeScript, e seu editor verifica a integridade do código o tempo todo —, o programa indicará que há um erro.

Caso façamos qualquer tipo de alteração, o projeto continuará com problemas no navegador. Neste caso, está sendo indicado que não é possível importar `PhotoComponent` em `AppModule`. Assim, deletaremos a linha `import { PhotoComponent } from './photo/photo.component'`, bem como `PhotoComponent` de `app.module.ts`.

Feito isso, já que ele não pertence a nenhum módulo, criaremos um `photos.module.ts` em "photos", que será o *feature module*. Nele, criaremos um módulo que irá declarar não só `PhotoComponent`, mas todos os outros componentes que dizem respeito a imagens, e `app.module.ts` importará este módulo nele mesmo.

Isso tornará a aplicação muito mais organizada. Em `photos.module.ts`, digitaremos "@NgModule" e pressionaremos "Enter" para fazermos a importação no `angular/core`, e entre parênteses passaremos um objeto JavaScript. E então declararemos `PhotoComponent`:

```
import { NgModule } from "@angular/core";

@NgModule({
    declarations: [ PhotoComponent ]
})
export class PhotosModule {}
```

COPIAR CÓDIGO

Se salvarmos o arquivo da maneira em que está e acessarmos o navegador, teremos uma página em branco, e no Console veremos que o componente não foi encontrado, pois ele precisa pertencer a um módulo. Assim, em `app.module.ts`, importaremos um módulo, e com isso teremos acesso aos componentes que foram declarados, automaticamente. Em `declarations`, só entram componentes, portanto colocaremos `PhotosModule` em `imports`:

```
@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        BrowserModule,
        PhotosModule
    ],
```

```
    providers: [],
    bootstrap: [AppComponent]
})
```

COPIAR CÓDIGO

Assim, quando o primeiro módulo da aplicação for carregado (`AppModule`), o programa entenderá que dependemos de `PhotosModule` , que será carregado, e por consequência o `PhotoComponent` terá que ser disponibilizado para uso.

Aproveitando, o que é que o Angular está importando em `BrowserModule` ? Este módulo traz vários recursos que utilizaremos ao longo do curso no navegador. Por isto, por padrão, o Angular CLI já faz sua importação.

Esta alteração que acabamos de fazer deve ser o suficiente para a nossa aplicação funcionar! Salvaremos, voltaremos ao navegador e... Continuamos com o mesmo erro. É indicado no Console que `ap-photo` não é um componente do Angular, e que precisamos verificar se ele faz parte de algum módulo.

Se criamos `photos.module.ts` e adicionamos `PhotoComponent` no `declarations` , como é possível que ele não faça parte de um módulo? Além disso, não acabamos de importá-lo em `app.module.ts` ?

Esquecemos de um passo importante: em `declarations` de `photos.module.ts` , se encontra tudo aquilo que o módulo possui. Caso tenhamos dez componentes, eles se enxergam entre si dentro do módulo. Mas para que ele seja enxergado no módulo de quem importou `PhotosModule` , precisaremos especificar na propriedade `exports` do `NgModule` , sendo necessário também torná-lo acessível para quem for importá-lo.

```
@NgModule({
    declarations: [ PhotoComponent ],
    exports: [ PhotoComponent ]
})
```

COPIAR CÓDIGO

Salvaremos e voltaremos ao navegador e, desta vez, tudo continua funcionando conforme esperado.

Entendemos que um módulo pode declarar muitos componentes, mas não exportar ou dar acesso a todos eles. Comparando com linguagens como Java ou C#, é como se tudo que se encontra em `declarations` estivesse privado, e em `exports`, público.