

Angular Material

Organização:

É muito bom organizarmos nossos imports principalmente os do **Angular Material** principalmente quando estamos trabalhando em uma aplicação, pois os módulos têm uma tendência de utilizar mais ou menos os mesmos componentes e termos todos os imports que a gente está utilizando em um módulo separado acaba deixando o código mais legível, porém não há problema se deixar todos os módulos no mesmo componente é só uma questão de legibilidade.

Para isso vamos criar um módulo para poder ter todos os imports do Angular Material disponíveis nesse módulo, se caso no futuro criarmos outros módulos para essa aplicação ao invés de importar tudo novamente (os imports) a gente só terá de importar o módulo do app material.

Como ele será um módulo compartilhado iremos gerar ele na pasta **shared** (Tudo que estiver nessa pasta será compartilhado com outros módulos da aplicação)

```
ng g m shared/app-material
```

Como não iremos usar esse módulo para criarmos componentes e nem organizar a aplicação, iremos usar ele apenas para poder economizar linhas de código no outros componentes que precisarão dele, tiraremos os **imports** do **@NgModule** e também tiraremos as **declarations**, pois não criaremos nenhum componente, mas no lugar iremos colocar o **exports**, ou seja iremos exportar todos esses módulos do Angular Material que tínhamos declarado até agora.

E **dentro do exports** colocaremos os módulos do Angular Material que importamos do nosso outro component no famoso ctrl+c e ctrl+v.

```
import { NgModule } from '@angular/core';
import { MatCardModule } from '@angular/material/card';
import { MatTableModule } from '@angular/material/table';
import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({
  exports: [
    MatCardModule,
    MatTableModule,
    MatToolbarModule
  ],
})
export class AppMaterialModule { }
```

E agora iremos importar esse modulo dentro do que estávamos utilizando e podemos nele remover tudo relacionado ao Angular Material e iremos apenas fazer o import do [AppMaterialModule](#). (Como mostra abaixo)

```

1 import { CommonModule } from '@angular/common';
2 import { NgModule } from '@angular/core';
3 import { AppModule } from '../shared/app-material/app-material.module';
4
5
6 import { CoursesRoutingModule } from './courses-routing.module';
7 import { CoursesComponent } from './courses/courses.component';
8
9
10 @NgModule({
11   declarations: [
12     CoursesComponent
13   ],
14   imports: [
15     CommonModule,
16     CoursesRoutingModule,
17     AppModule
18   ]
19 })
20 export class CoursesModule { }

```

E usando esse método temos a facilidade de sempre que formos adicionar um novo componente do Angular Material iremos adiciona-lo no `AppMaterialModule`.

Table:

`th`: parte do título

`td`: para cada coluna que iremos ter na nossa tabela

`ColumnsToDisplay`: iremos colocar todas as colunas que queremos mostrar.

Spinner:

Quando o usuário entra em nossa página e precisamos carregar algum dado pode acabar demorando para eles chegarem do nosso servidor até o nosso **front-end** e é interessante o **usuário** saber que está sendo carregado os dados da página em questão, para isso temos no **Angular Material** o **spinner** ([Progress spinner](#)) que é a "**bolinha**" de carregamento que vemos nos sites para informar ao usuário que tem algo sendo carregado.

Para fazer a lógica de quando mostrar o **spinner** e quando mostrar a tabela no nosso **HTML** iremos envolver a **table** (tabela) em uma **div** e nela usaremos o **ngIf** com a condição de: se tivermos os dados mostrarmos a tabela e se não tivermos os dados mostraremos a "bolinha" de carregamento até ter eles, porém para verificarmos se os nossos dados foram carregados usaremos o **ngIf** no **courses** que é um **observable** e temos de usar o **pipe async** que temos no Angular, pois este **pipe async** ele irá automaticamente se inscrever no **observable** e ao deixarmos de usar esse componente ou muda a rota da página e esse componente é destruído automaticamente pelo Angular, o Angular irá fazer o **unsubscribe** automaticamente, mas no nosso caso isso não importa muito por estarmos usando o operador **first** que automaticamente pega a primeira resposta fornecida e encerra a conexão, como usaremos essa informação depois iremos dar a ela um nome.

```

<mat-card>
  <mat-toolbar color="primary">Cursos Disponíveis</mat-toolbar>

  <div *ngIf="courses$ | async as courses; else loading">
    <table mat-table [dataSource]="courses" class="mat-elevation-z8">
      <!-- Name Column -->
      <ng-container matColumnDef="name">
        <th mat-header-cell *matHeaderCellDef>Curso</th>
        <td mat-cell *matCellDef="let course">{{ course.name }}</td>
      </ng-container>

      <!-- Category Column -->
      <ng-container matColumnDef="category">
        <th mat-header-cell *matHeaderCellDef>Categoria</th>
        <td mat-cell *matCellDef="let course">{{ course.category }}</td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
      <tr mat-row *matRowDef="let row; columns: displayedColumns"></tr>
    </table>
  </div>
  <ng-template #loading>
    <div class="loading-spinner">
      <mat-spinner></mat-spinner>
    </div>
  </ng-template>
</mat-card>

```

Ali no ng-template #loading o loading da o nome aquele ng-template e é usado no else da div.