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.

```ts
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)

```ts
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { AppMaterialModule } from '../shared/app-material/app-material.module';


import { CoursesRoutingModule } from './courses-routing.module';
import { CoursesComponent } from './courses/courses.component';


@NgModule({
  declarations: [
    CoursesComponent
  ],
  imports: [
    CommonModule,
    CoursesRoutingModule,
    AppMaterialModule
  ]
})
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 titulo

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.

Mat-header-cell:

No Angular Material nós temos um componente chamado Mat-header-cell que nele conseguimos configurar exatamente como queremos customizar nossa coluna, é um pouco diferente de utilizar um TH, TD por exemplo, pois neles temos um texto e dados que estão vindo da nossa coleção respectivamente.

```
<ng-container matColumnDef="category">
  <th mat-header-cell *matHeaderCellDef>Categoria</th>
  <td mat-cell *matCellDef="let course">{{ course.category }}
    <mat-icon aria-hidden="false" aria-label="Categoria do curso">{{ course.category | category }}</mat-icon>
  </td>
</ng-container>
```

Vamos então usar a diretiva *matHeaderCellDef no mat-header-cell, essa diretiva irá falar para o Angular que esse é o cabeçalho, após adicionar um botão e o ícone dele

```
<ng-container matColumnDef="actions">
  <mat-header-cell *matHeaderCellDef>
    <button mat-mini-fab color="primary" aria-label="Adiconar um curso">
      <mat-icon>add</mat-icon>
    </button>
  </mat-header-cell>


</ng-container>
```

Temos que adicionar no displayedColumns já que é ele quem renderiza as coisas na nossa página o component só declara, veremos após isso que será gerado um erro no nosso console e isso é porquê para essa coluna a gente apenas definiu o nosso cabeçalho (header), está faltando o corpo agora, enquanto não definirmos o corpo a gente não vai ver os dados e terá esse erro também, então no Material Table a gente não consegue ter apenas o cabeçalho sem ter o conteúdo dessa coluna.

E para definirmos o corpo usaremos o mat-cell.

Vamos utilizar esses 2 (mat-header-cell, mat-cell) ao invés do TD e TH, pois sempre que quisermos ter mais customização, ter mais controle do que o Angular vai renderizar para a gente no Material Table a gente utiliza eles.

Mat-cell:

No mat-cell iremos colocar a diretiva *matCellDef com o valor let course, fazendo com que para cada registro que vier da nossa coleção iremos fazer alguma coisa com esse registro.

E nesse caso colocaremos nossos botões de editar e remover nele.

Validators:

Temos no Angular a classe Validators que tem algumas validações prontas e usamos ela nos nossos atributos que queremos essas validações.

Validators:

Temos no Angular a classe Validators que tem algumas validações prontas e usamos ela nos nossos atributos que queremos essas validações.

Validators:

Temos no Angular a classe Validators que tem algumas validações prontas e usamos ela nos nossos atributos que queremos essas validações.

```
name: [
  '',
  [Validators.required, Validators.minLength(3), Validators.maxLength(100)],
],
category: ['', [Validators.required]],
```

Porém essa classe faz a validação visual, ou seja ela identifica o campo com algum erro e fica vermelho o local que não passou na validação, porém não

impede do erro seguir, como exemplo podemos ainda sim salvar cursos com nome e categoria vazio.