**Angular - Angular material**

Para adicionar o Angular material: cmd na pasta do projeto: ng add @angular/material

Componentes:

Toolbar: copiar a API do componente do appmodule





Texto

Descrição gerada automaticamente



Extensão para organizar os imports: TS Hero. View – command palette (ctrl+shift+p).

Colar a diretiva toolbar no app componente.html

<mat-toolbar color="primary">

  <span>Projeto Sumô</span>

</mat-toolbar>

<router-outlet></router-outlet>

Criar o módulo rikishi com o roteamento: ng g m rikishi --routing

Todos os componentes criados no módulo só são visíveis dentro dele.

Criar o componente rikishi: ng g c rikishi/rikishi

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Declarar as rotas no rikishi-routing-module:

const routes: Routes = [

  { path: '', component: RikishiComponent }

];

Declarar as rotas no app-routing-module

{ path: '', pathMatch: 'full', redirectTo: 'rikishi' },

  {

    path: 'rikishi',

    loadChildren: () => import('./rikishi/rikishi.module').then(m => m.RikishiModule)

  }

Import do html5 de forma dinâmica:

./ faz referência ao app. O caminho será onde está o módulo filho (rikishi.module).

m. faz referência a qual modulo será carregado.

Se quiser adicionar mais módulos é só colocar aqui.

Quando clicar no endereço /rikishi é que o angular vai carregar esses arquivos declarados aqui. Mas aqui como foi feito redirecTo: rikishi ele vai carregar de qualquer forma.

Cores:

<https://material.io/design/color/the-color-system.html#tools-for-picking-colors>

no style.css global:

//imports

@import '@angular/material/theming';

@include mat-core();

//cores que serão usadas

$custom-app-primary: mat-palette($mat-pink); //cor primária

$custom-app-secundary: mat-palette($mat-blue-gray, 600, 700, 800); //cor secundária

$custom-app-warn: mat-palette($mat-red); //alerta

//cor do tema

$custom-theme: mat-light-theme($custom-app-primary, $custom-app-secundary, $custom-app-warn);

  @include angular-material-theme($custom-theme);

Componente para listar os rikishis:

Table: copiar a API no rikishi.module

Texto, chat ou mensagem de texto

Descrição gerada automaticamente



Copiar a diretiva e colar no rikishi.html

Tela preta com letras brancas

Descrição gerada automaticamente...

Para o Datasource:

Criar um folder na pasta rikishi com o nome model e no cmd: ng g interface rikishi/model/rikishi (assim já cria a interface com o nome dentro da pasta).

Na interface:

Texto

Descrição gerada automaticamente

No controller do rikishi:

Adicionar o array do rikishi model

Texto

Descrição gerada automaticamente

Obs: OnInit: faz parte do ciclo de vida de um componente.

E no html alterar o datasource para rikishi e preencher o restante:

Obs: datasource pode ser um array ou um observable.

Texto

Descrição gerada automaticamente

No controller:

Interface gráfica do usuário, Texto, Site

Descrição gerada automaticamente

Adicionar uma linha para verificar se funcionou:

Tela preta com letras brancas

Descrição gerada automaticamente

Melhorando o css no componente:

Colar o css da table do site do angular

Texto

Descrição gerada automaticamente

Adicionando card:

Colar a api do card no rikishi.module, e então copiar o código do card no html envolvendo a tabela

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente



Adicionar uma toobar (importar api no modulo rikishi desta vez e colar código no html).

Texto

Descrição gerada automaticamente

Criar um modulo para colocar todos os imports do angular

ng g m shared/app-material

Copiar os imports do rikishi module para este arquivo e adicionar no ngModulo um exports com os nomes: ficara assim:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

No rikishi module: apagar os imports e adicionar o AppMaterialModule.

Texto

Descrição gerada automaticamente

Criar o serviço que conterá a lógica: ng g s rikishi/services/rikishi

O componente em si não sabe quais dados serão renderizados, quem passa isso para ele é o serviço. A única coisa que o componente sabe é que ele vai renderizar uma lista de rikishis, de onde isso está vindo é irrelevante. Essa info é abstraída do componente.

Texto

Descrição gerada automaticamente



Injectable: vai fazer parte da injeção de dependência das classes dentro do angular.

Para usar os dados da API que foi construída em Java (no spring) é necessário fazer uma chamada Ajax (chamada assíncrona com o servidor). No angular a classe que fornece os métodos para isso é o HttpClient.

Importar a classe HttpClient e escrever no construtor.

Ao declarar no construtor ocorre a injeção de dependência (angular: instancie o httpClient para mim - essa instância será fornecida na raiz do projeto (provideIn: ‘root’) ou seja, será global.

Criar o método que vai listar os rikishis (para isso a url da api deve estar no environment). Metodo list: o retorno do método get é um observable e no get usa-se o operador de diamante<> para indicar qual o tipo de objeto que vamos receber do servidor. Alterar no controller também a variavel rikishi para observable e para inicializar colocar no controlador a chamada do metodo.

Para debugar: colocar um pipe pra poder manipular a info que virá e um console log dentro de um tap.

First(): obtem a prmeira info vinda do servidor e finaliza a inscrição do rxjs no endopoint.

Tela preta com letras brancas

Descrição gerada automaticamente

Adicionar o HttpClientModule no app.module

No controller:

Texto

Descrição gerada automaticamente



No componente a injeção do serviço é feita no construtor. Quando eu declaro a variável rikishiService no construtor ela se torna uma variável da instancia, ou seja, do componente. Para carregar o método somente quando o componente foi inicializado, adicionar o método no ngOnInit() se não, colocar no construtor.

Spinner:

Copiar a api do material.io no app-material.module (criado por nós para colocar todos os imports).

Antes de colar a diretiva do spinner, no html: envolver a table em uma div. Fazer um ngIf para que SE não estiver retornando dados ativar o spinner.

<div \*ngIf="rikishi$ | async as rikishi; else loading">

Atenção: utilizar o | async pq automaticamente ele se inscreve no observable rikishi e assim que o angular destrói esse componente o angular faz o unsubscribe automático. (colocar no observable do controller o sinal de dólar por convenção). E como vamos utilizar essa variável depois, damos o nome de rikishi. +atenção: o datasource agora não faz mais referência ao observable e sim a variável rikishi do async.

Aqui, o comando ngIf: se inscreve no observable, extrai as info, atribui as info para a variável local rikishi e a tabela usará essas info da variável para renderizar tudo.

Ao colocar o else, fazemos referencia a um outro template (loading – nome que eu dei)que existe no html. Essa condição só será feita quando a primeira não for satisfeita.

Adicionar um ng template com o identificador loading, colar a diretiva do spinner dentro do ng-template.

Texto

Descrição gerada automaticamente

Para testar o spinner: ir no serviço – delay(5000) = 5segundos

Tratamento de erros:

ng g m shared – criar modulo shared (que envolvera o app-material)

ng g c shared/components/error-dialog – criar pasta dentro do modulo

Quem trata o erro é o componente – através do pipe - : rikishi.ts

abrir um pipe e escrever o return que espera um observable nós podemos criar um observable de qualquer informação estática que está na memória com o operador of (rxjs) que significa: retornar um array vazio para que o spinner pare de funcionar.

//código pronto

Texto

Descrição gerada automaticamente

Mas é interessante mostrar ao usuário que ocorreu um erro, para isso:

Para separar a parte de erro:

ng g m shared – criar modulo shared (que envolvera o app-material)

ng g c shared/components/error-dialog – criar pasta dentro do modulo

como a pasta de módulo shared será usada no componente rikishi é preciso exportá-la:

Texto

Descrição gerada automaticamente



Importar o modulo app-material.module no shared module (módulos são importados, componentes são declarados!)

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente



Para escrever o catchError dentro do pipe:

Procurar no material uma dialog

Colar a API no app-material

Abrir o controller do error e colar o construtor do material:

Texto

Descrição gerada automaticamente



Fazer os imports necessários e alterar o data para string pq será uma mensagem que será passada:



Ou seja, vamos injetar o tipo mat dialog data na data: string.

Para linkar: colocar no html um p com {{ data }} depois será alterado.

No controler rikishi copiar o metodo openDialog do material (seta indicando foto acima) e renomear para onError, adicionar o parâmetro string

Texto

Descrição gerada automaticamente

Injetar no construtor o matdialog (asterisco) depois do rikishiService e fazer o import.

Texto

Descrição gerada automaticamente

Fazer o import do shared.module para o rikishi.module

Texto

Descrição gerada automaticamente



Com isso, podemos adicionar o ErrorDialogComponent dentro do open (método onError) e mudar o data:

Texto

Descrição gerada automaticamente



Voltando ao pipe – catch error:

Adicionar a mensagem que o usuário verá qdo o método onError for chamado, ficando assim:

Texto

Descrição gerada automaticamente

Se tudo der certo, estará funcionando.

Melhorando o popup

Copiar o html da dialog elements

Texto

Descrição gerada automaticamente

E substituir no data do html

Texto

Descrição gerada automaticamente

Estilizar com cor e alinhamentos.

Para o botão:

Copiar a api no app material e alterar para mat-stroked-button (tipo de botão).