***VALIDACAO DE FORMULARIOS COM NGMODEL***

ngform -> para validar o form

Para utilizarmos o templete forms é necessário a diretiva ngModel e o atributo name

Form rastreia os campos com o NGMODEL de forma que se os campos com NGMODEL forem inválidos o form também será invalido.

NGFORM nos permite mudar as propriedades de um elemento no input por meio do [(ngModel)]. "TWO-WAY BINDING" -> (input e output) ||||| "ONE-WAY BINDING" -> (output)

#myForm="ngForm" nos permite acessar as propriedades do formulario.

Entao podemos por exemplo: habilitar ou não um botão baseando-se na validade do form.

Gerando componente: ng g c order --spec=false

NGMODEL: ESTADOS DOS CAMPOS

VALID|INVALID

PRISTINE / DIRTY -> O campo se encontra no primeiro estado, assim que o usuário digita ele passa a ser DIRTY

TOUCHED -> quando o usuário entra no campo.

Para saber o estado do compo precisamos de utilizar o \*ngIf="ipt.invalid"

***VALIDATORS***

Validamos os campos de um formulário criando um componente chamado mt-input-container e um input como filho dele.

A partir da classe has-error e has-success do bootstrap damos uma aparencia melhor para as validações.

***VALIDACAO COM EXPREÇÕES REGULARES***

Para validarmos com EXPREÇÕES regulares devemos usar o atributo pattern="" no input que queremos validar

Expressões regulares usadas na validação de formulários

### Email Regex

/^(([^<>()\[\]\.,;:\s@\"]+(\.[^<>()\[\]\.,;:\s@\"]+)\*)|(\".+\"))@(([^<>()[\]\.,;:\s@\"]+\.)+[^<>()[\]\.,;:\s@\"]{2,})$/i`

### Number Regex

`/^[0-9]\*$/`

***COMPONENTE PARA Radio - ControlValueAccess***

Um componente foi criado para representar o radio, através de um array controlamos os elementos do radio e pagamos o value digitado pelo

Evento de click

***REACTIVE forms***

O form é representado por um form group userForm: FormGroup (interface)

Declaramos um private fb: FormBuider para inserir os campos de nosso form.

É equivalente ao templete-form

Dá para validar valores de grupos, com isso conseguimos validar os valores entre os componentes.

***LAZY LOAD ou Carregamento tardio***

Com Lazy load conseguimos carregar a página somente no momento da requisição a ela, por exemplo, em funcionalidades que dependem de permissões especiais. Não a necessidade carrega-la antecipadamente.

Carregar um determinado modulo somente por necessidade. Sem LAZY LOAD a aplicação carrega todos os componentes. Em aplicações maiores isso pode acarretar em lentidão em nossa SPA.

Por exemplo o componente que representa o sobre da nossa aplicação é um bom concorrente para virar um modulo. Pois, nem todo mundo que acessa a aplicação ira para o sobre.

***BOAS PRATICAS - MODULARIZAÇÃO DE COMPONENTES COMPORTATILHAVEIS***

Input - radio - rating - entre outros components

***CORE MODULES***

Sempre que colocamos um serviço na nossa lista de providers, nós estamos criando uma instancia daquele serviço e disponibilizando para os componente e para seus filhos.

A injeção de dependencia é feita de forma Hierarquica.

O ponto mais alto da herança fica em um ponto que é chamado de módulo raiz. "app.module.js".

Quando criamos um Modulo separado, como o orderModule os providers declarados dentro deste moludo não será visivel para o restante dos modulos da aplicação.

Neste ponto entra o CORE Modulo, ele tem como ideia tratar este problema, ou seja, nós teremos um modulo separado onde temos todos os providers da aplicação e esse modulo

só é importado no modulo raiz, evitamos então ter uma instancia a mais de nossa injeção de dependencia.

**Modulo - ModuleWithProviders**

Nos permite importar o modulo de forma diferente, com providers ou sem providers.

***PRÉ-CARREGAMENTO DE MODULOS***

Com o PreloadAllModules nós conseguimos carregar os módulos configurados para carregamento tardio em modo background,

Sem o PreloadAllModules podemos ter lentidão na hora de carregar módulos tardios. Já com ele tudo será carregado no background

Melhorando a experiência da aplicação.

***ANIMAÇÕES COM ANGULAR*** – <https://www.w3schools.com/css/css3_transitions.asp>

As animações com angular nos tras uma ideia de movimento a aplicação. Ela está disponível somente para o firefox e google chrome.

Para os navegadores que não suporta temos que importar o polifill.

1ºimportando o modulo de animações.

npm install --save @angular/animations@4.0.0

2º importando o polifill para que a aplicação rode em qualquer navegador.

npm install --save web-animations-js

3º configurar o nosso polifill

4º configurar o nosso animations module no nosso modulo principal.

***IMPLEMENTANDO UM SERVIÇO DE NOTIFICAÇÕES PARA NOSSA SNACKBAR***

Criamos um serviço que emite um evento para a aplicação:

notify(message: string) {

this.notifier.emit(message)

}

Assim ele ficara disponivel para outros pontos da aplicação...

Para pegar este evento: Precisamos injetar este serviço no componente onde queremos importa-los.

Vamos adiciona-lo em dois pontos.

No shopping-cart.service.ts temos dois pontos legais onde podemos chama-lo, ao inserir itens e remover itens do carrinho.

this.notificationService.notify(`Você adicionou o item ${item.name}`)

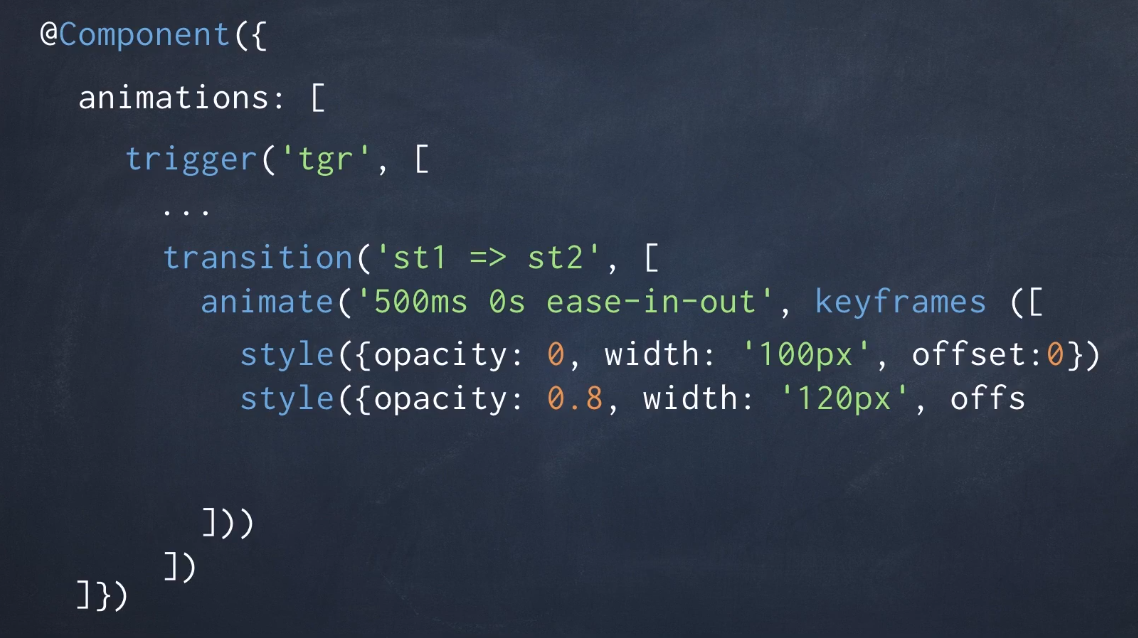
***IMPLEMENTANDO UM UM TIMER PARA NOSSA SNACKBAR***

Quando ativamos uma snackbar para hiden ela não desaparece mais. Para que ela desapareça utilizamos um operador chamado timer do Observable.

Observable.timer(3000).subscribe(timer => this.snackVisibility = 'hidden')

***ANIMAÇÕES COM KEYFRAME*** |

Mudar as animações em pontos diferentes através do keyframes.



***CRIANDO ROTA PARA PAGINA NÃO ENCONTRADA***

Basta criar uma rota \*\* e adicionar um componente que represente está pagina.