Validação com expressões regulares:

Expressões regulares seriam um conjunto de caracteres, a sua validação seria impedir que esses caracteres seja válidos. Por exemplo:

            <input type="text" class="form-control" pattern="^[0-9]\*$" name="number" ngModel required placeholder="Número" autocomplete="off">

Para isso utilizamos o pattern e passamos o que não queremos que seja válido no campo.

ControlValueAccessor: Uma interface capaz de dar poderes aos componentes no sentido que pode se ter uma ponte para conversar com as diretivas especiais de formulário como ngModel ou as diretivas de reactive forms.

Para de utilizar essa interface primeiro importamos ela:

import {ControlValueAccessor} from '@angular/forms';

Depois implementamos a nossa classe:

export class RadioComponent implements OnInit, ControlValueAccessor {

Para se utilizar a interface é necessária a implementação de alguns métodos:[

Sendo os mais utilizados:

 /\*\*

     \* Write a new value to the element.

     \*/

    //é um método que vai ser chamado pelas diretivas quando elas querem passar um valor para o componente

    writeValue(obj: any): void{

      this.value = obj;

    };

    /\*\*}

     \* Set the function to be called when the control receives a change event.

     \*/

    //é passada uma função pra gente, que a gente tem que chamar essafunção sempre que o valor, sempre que o valor interno do componente mudar.

    //então sempre que a função mudar temos que chamar a var onChange passando o valor

    registerOnChange(fn: any): void{

        this.onChange = fn;

    }

E também possuem os :

  /\*\*

     \* Set the function to be called when the control receives a touch event.

     \*/

     //usado caso queira registrar que o usuário entrou nos eu componente

    registerOnTouched(fn: any): void{

    }

    /\*\*

     \* This function is called when the control status changes to or from "DISABLED".

     \* Depending on the value, it will enable or disable the appropriate DOM element.

     \*

     \* @param isDisabled

     \*/

     //

    setDisabledState?(isDisabled: boolean): void;

Para utilização do ValueAccessor é necessário registrá-lo na lista de providers, porém esse registro é feito no próprio componente, o provider utilizado é o NG\_VALUE\_ACCESSOR:

//Registrando o componente como um valueAccessor

import {NG\_VALUE\_ACCESSOR, ControlValueAccessor} from '@angular/forms';

@Component({

  selector: 'mt-radio',

  templateUrl: './radio.component.html',

  styleUrls: ['./radio.component.css'],

//registrando provider no próprio componente

  providers: [

    {

      provide: NG\_VALUE\_ACCESSOR,

//fazendo referencia ao componente que criamos

 useExisting: forwardRef(()=> RadioComponent),

      multi: true

    }

  ]

})

Dessa forma o componente fica disponível para ser usado com as diretivas do ngModel e reactivesforms que ainda seram vistas.

Transformando tipos do item:

As vezes um objeto é declarado de um certo tipo, e no meio do caminho precisa ser atribuído a um outro tipo, é possível fazer isso com map:

order.orderItem =  this.cartItems().map((item:CartItem) =>

    new OrderItem(item.quantity, item.menuItem.id));

O retorno de cartItems() era do tipo CartItem, e nesse momento precisava que fosse OrderItem, então é feita essa conversão com map.

Sempre que se faz um post é necessário adicionar algumas coisas no header, por padrão sempre adicionamos:

 const header = new Headers();

         header.append('Content-type', 'apliccation/json');