**Angular Tutorial:**

**Validar Form**

1. **Criar projeto**
2. Instalar o angular cli : npm install –g @angular/cli
3. Criar diretório projeto : ng new <nome do projeto>
4. Incluir as dependências no projeto: npm install
5. **Configurando as dependências**
6. Vamos incluir as dependências no file app.module.ts
7. HttpModule : O módulo que inclui provedores de http
8. FormsModule :
9. ReactiveFormsModule:

Esses módulos devem estar no campo “imports”

1. Todo componente introduzido no projeto deve ser importado e incluso o file app.module.ts . Todo componente deve estar no campo “declarations”
2. **Criando um componente**
3. Comando por angular cli : ng g c <nome\_do\_componente>
4. O componente de conter os módulos FormGroup,FormBuilder,Validators e FormControl
5. **Agrupando elementos no Form**
6. O componente deve conter um objeto que instancie o FormGroup

Exemplo:

**private formLogin: FormGroup;**

**private validarForms() {**

**this.formLogin = this.formBuilder.group({**

**campo\_para\_validar: ['', [CustomValidators.required]]**

**});**

**}**

1. O construtor do componente deve conter a injeção do modulo FormBuilder

Exemplo:

**constructor(private formBuilder: FormBuilder){}**

1. O componente deve iniciar o método **validarForms()**
2. **Incluindo campos do Formulário**
3. O html relacionado a componente deve conter a tag form
4. A tag Form deve conter a propriedade **formGroup** que receberá  **formLogin**
5. A tag Form deve conter **ngSubmit** que receberá o método **metodoSend()**

Exemplo:

**<form [formGroup]="formLogin" (ngSubmit)="metodoSend()">**

1. Para a validação de um campo devemos indicar as seguintes propriedades

**[(ngModel)]="campo\_para\_validar"**

**formControlName="campo\_para\_validar"**

1. **Validando campos**
2. Vamos criar um método responsável pela validação **isInvalid(field: string, validate: string)**

O parâmetro **field** corresponde ao campo(**campo\_para\_validar**) que incluímos no html.

O parâmetro **validate** corresponde ao tipo (**required**) relacionado ao **field** .

Como a tag input está contida na tag form podemos manipular o campo com o seguinte código: **const fieldControl = this.formLogin.controls[field];**

Com esse resultado podemos saber se o campo foi selecionado **(fieldControl.touched)** ou se o campo está valido **(fieldControl.invalid)**

Desta forma teremos o seguinte resultado:

**isInvalid(field: string, validate: string) {**

**const fieldControl = this.formLogin.controls[field];**

**return (fieldControl.touched) && (this.formLogin.hasError(validate)**

**|| ((validate) ? (fieldControl.hasError(validate)) : fieldControl.invalid));**

**}**

1. Vamos incluir o trecho abaixo no html

**<input**

**type="text"**

**id="campo\_para\_validar"**

**name="campo\_para\_validar"**

**[(ngModel)]="campo\_para\_validar"**

**formControlName="campo\_para\_validar"**

**[class.erro]="isInvalid('campo\_para\_validar')">**

1. Para podermos manipular o input devemos incluir uma propriedade no file .ts correspondente ao componente

**private campo\_para\_validar: string;**

Observe que a propriedade possui o mesmo nome indicado em

**[(ngModel)]**

1. **Validando o Form**

Para validar o form é necessário incluir um método que instancie o objeto **FormBuilder** e inclua o campo que queremos validar. Para isso vamos agrupar o(s) campo(s) utilizando o método group conforme exemplo abaixo:

**private validarForms() {**

**this.formLogin = this.formBuilder.group({**

**campo\_para\_validar: ['', [Validators.required]]**

**});**

**}**

Observe que aqui indicamos que o campo **(campo\_para\_validar)** é requerido.

1. **Submetendo o form**

Para Submeter o from vamos incluir a tag button no interior da tag form

**<button type="submit" >Enviar</button>**

O evento click do button corresponde ao método **(ngSubmit)="metodoSend()"** indicado no topo da tag form.

Este evento no file .ts validará o formulário como no exemplo abaixo.

**metodoSend() {**

**if (this.formLogin.valid) {**

**this.mensagem = 'Form OK';**

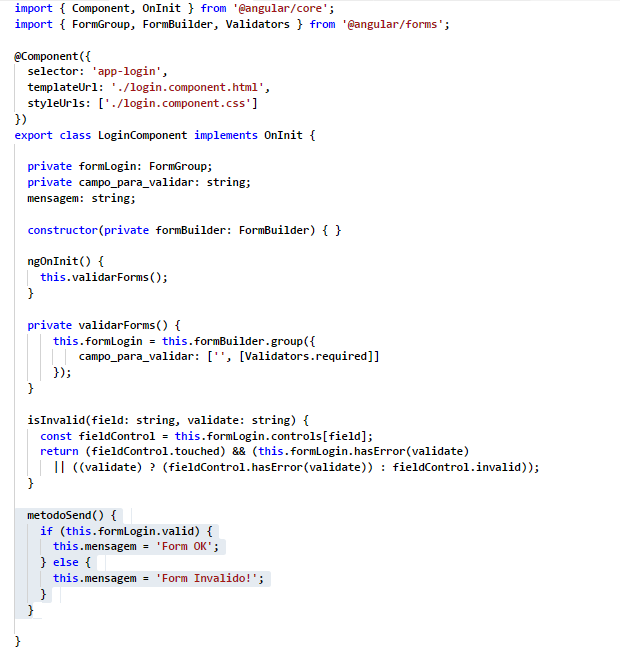
**} else {**

**this.mensagem = 'Form Invalido!';**

**}**

**}**

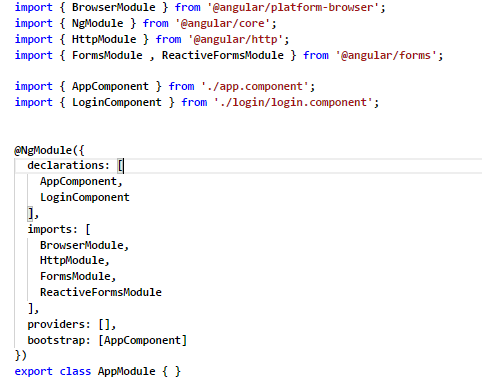
**src\app\login\login.component.ts**



**src\app\login\login.component.html**



**src\app\app.module.ts**



**Usando Input, Output e EventEmitter**

Input @angular/core : Passar dados de pai para filho

****

Output @angular/core : O pai escuta o evento filho

O componente filho expõe uma propriedade EventEmitter com a qual ele emite eventos quando algo acontece. O pai se liga a essa propriedade do evento e reage a esses eventos.

A propriedade EventEmitter do filho é uma propriedade de saída, geralmente decorada com uma decoração @Output.



**Criar um serviço usando Observable**

Observable emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disso podemos executar uma ação.

Principais Imports:

**import {Observable} from 'rxjs/Observable';**

**import 'rxjs/add/operator/catch';**

**import 'rxjs/add/operator/map';**

O método relacionado seria

**GetTagAlizeObservable(text: string): Observable<Tag[]> {**

**return this.http.get(`${this.url}Util/?text=${text}`)**

**.map( (result) => result.json())**

**.catch(erro => Observable.throw(erro.message));**

**}**

**Configurar map para models**

Para configurar o mapping de uma model é necessário utilizar o método **Object.assign()** nativo do javascript.

O método **Object.assign()** é usado para copiar os valores de todas as propriedades próprias enumeráveis de um ou mais objetos *de origem* para um objeto *destino*. Este método irá retornar o objeto *destino*.

Object.assign(*destino*, ...*origens*)

Vamos criar uma class contendo em seu construtor a chamada de método

**export class Model {**

**constructor(objeto?) {**

**Object.assign(this, objeto);**

**}**

**}**

Esta class será herdada pelas demais models

Exemplo:

**import { Model } from '../model';**

**export class Tag extends Model {**

**constructor(obj?: any) {**

**super(obj);**

**}**

**public id: number;**

**public simpleText: string;**

**public normalized: string;**

**public idSubjectMatter: number;**

**}**

Como o método **Object.assign()** possui os parâmetros (objetos *de origem* ) e (objeto *destino* ) . Como podemos ver **Object.assign(this, objeto);** o tremo **this** referece a class que está herdando e o termo **objeto** referece ao valor que queremos mapear.