Ojo a los símbolos que Word los cambia

# Comandos

## Crear proyecto

* Ng new [miProyecto]

## Servidor

* Ng serve: arranca el servidor en un puerto
* Ng serve --open: arranca el servidor y abre el navegador con el puerto del server

## Componente

* Ng generate component [miComponente]: genera un componente con ese nombre
* Ng g c [miComponente]: lo mismo que el anterior pero abreviado.

## Servicio

* Ng generate service [miServicio]: genera un servicio
* Ng g s [miServicio]: abreviatura

## Importar proyecto

* Npm install: descarga librerías requeridas por el proyecto importado

# DOM Angular

En todo el proyecto solo puede haber una vez cada una de estas etiquetas: Doctype html, head y body. Hay que colocarlas así:

* Index.html: Es el fichero base del html
  + <Doctype html><head>blablablá</head><body>(header)(navBar)<app-component></app-component>(footer)</html>.
* App.component.html: Este es el componente principal y es el que se usa para ir mostrando todas las páginas de nuestra web. La mecánica es que cada página sea un componente y cuando estemos en esa página se active el componente de esa página en el app.component.html:

# Bootstrap

* Comando: npm i Bootstrap
* Angual.json:

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

"root": "",

      "sourceRoot": "src",

      "prefix": "app",

      "architect": {

        "build": {

          "builder": "@angular-devkit/build-angular:browser",

          "options": {

            "outputPath": "dist/proyecto",

            "index": "src/index.html",

            "main": "src/main.ts",

            "polyfills": "src/polyfills.ts",

            "tsConfig": "tsconfig.app.json",

            "inlineStyleLanguage": "scss",

            "assets": [

              "src/favicon.ico",

              "src/assets"

            ],

            "styles": [

              "src/styles.scss",

              "node\_modules/bootstrap/dist/css/bootstrap.min.css"

            ],

# Componentes

Comando “ng g c [nombre\_componente]”

# RouterLink

Se usa para poder navegar entre las páginas con los <a href>

## router-outlet

Se añade la etiqueta <router-outlet></router-outlet> en el fichero app.component.html

<div class="background">

    <app-navbar></app-navbar>

    <router-outlet></router-outlet>

</div>

<app-footer></app-footer>

<router-outlet> es el componente donde va a aparecer todo el contenido al que naveguemos.

## Routing.module

En el fichero app-routing.module.ts que está dentro de la carpeta app.

Aquí importaremos los componentes a los que nos vamos a poder mover y en routes colocamos el path que queramos para cada componente. Este path es el que usaremos en los .html en la etiqueta <a>.

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { AnalisisSentimientosComponent } from './components/analisis-sentimientos/analisis-sentimientos.component';

import { DescriptionComponent } from './components/description/description.component';

import { FooterComponent } from './components/footer/footer.component';

import { NavbarComponent } from './components/navbar/navbar.component';

import { WelcomeComponent } from './components/welcome/welcome.component';

const routes: Routes = [

{

path:'analisis-sentimientos', component: AnalisisSentimientosComponent

},{

  path:'', component: WelcomeComponent

},{

  path:'description', component: DescriptionComponent

},

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

## [routerLink]

En los .html de los componentes colocaremos el atributo [routerLink] en la etiqueta <a [routerLink]=”[‘/path’]”></a>

<a [routerLink]="['/description']"><button><b>Introducción al Análisis de Sentimiento</b></button></a>

## RouterLinkActive

Hay que poner la propiedad routerLinkActive en la etiqueta:

<a class="nav-link" aria-current="page" [routerLink]="['']" routerLinkActive="activa" [routerLinkActiveOptions]="{exact: true}">Página de inicio</a>

activa es la clase CSS que se aplicará cuando esté activa la ruta.

[routerLinkActiveOptions]="{exact: true}" es para cuando se quiere que la ruta sea exactamente igual a la que se ha puesto en routerLink, básicamente para mayúsculas, minúsculas y los parámetros que lleve la ruta.

## Router.Navigate

Se usa para redirigir a otras páginas desde los .ts

### Para QueryParamas

Se pone la ruta incluyendo los PathParams y se añaden los QueryParams en un json de nombre “queryParams”:

this.router.navigate(['/busqueda/PathParams'], {queryParams:{"municipio":texto}})

# Crear API con Python Flask



## Importar e instalar flask

Se instala flask que viene con todas las librerías que necesita:

from flask import Flask, jsonify, request, abort, make\_response #pip install Flask

### Para solucionar el error de CORs:

from flask\_cors import CORS, cross\_origin #CORs $pip install -U flask-cors

#Bucle del servidor

app = Flask(\_\_name\_\_)

CORS(app)#CORs

## GET

La primera línea es para crear el end-point.

<int:id> es un parámetro de tipo entero con nombre ‘id’.

@app.route('/actividades/<int:id>', methods = ['GET'])

def get\_actividad(id):

    actividad = list(filter(lambda t: t['id'] == id, actividades))

    if (len(actividad)==0):

        return make\_response(jsonify({ 'error': 'Actividad no encontrada'}), 404)

    return jsonify({'actividad': actividad[0]})

## Post

@app.route('/actividades', methods = ['POST'])

def create\_actividad():

    if not 'titulo' in request.json:

        abort(404)

    actividad = {

        'id' : actividades[-1]['id'] + 1,

        'titulo': request.json['titulo']

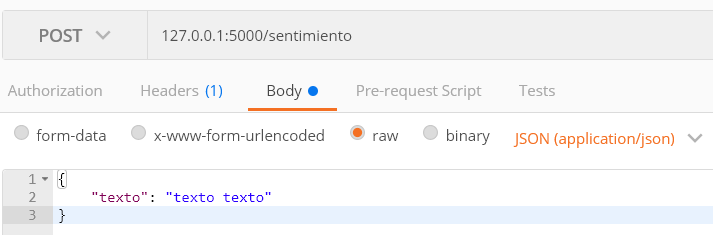
    }

    actividades.append(actividad)

    return jsonify ( { 'actividad': actividad })

## Postman

Herramienta para probar los end-points:



Instalárselo con Chrome no requiere cuenta.

# Service

## Buenas prácticas

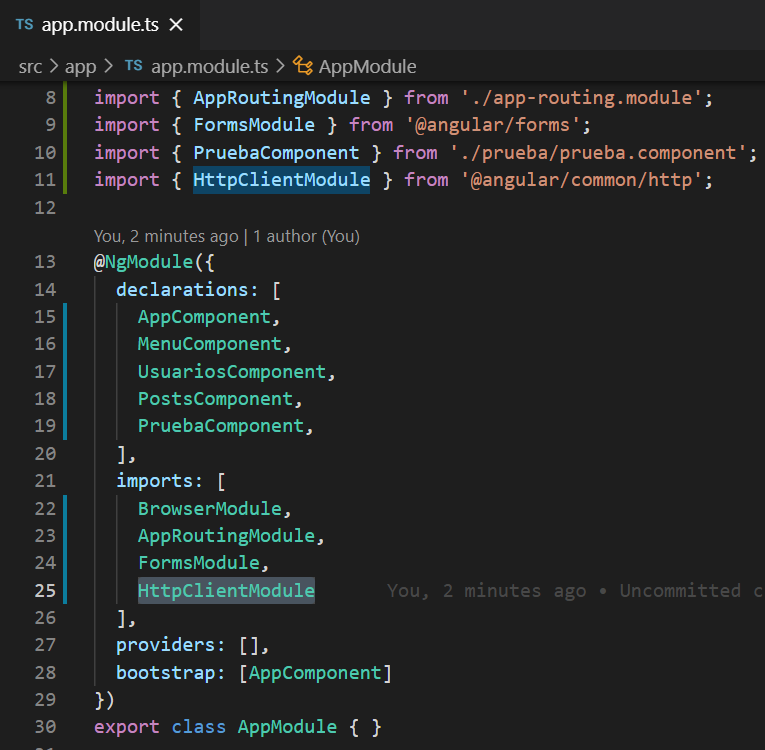
Crear los servicios en la carpeta src/app/services.

Los nombres todo\_en\_minusculas\_y\_guiones\_bajos. Que angular hace lo que le da la gana con ellos.

La transformación a objeto json se hace en el .ts del service.

## HttpClientModule

Para la comunicación http hay que importar HttpClientModule.



## mi\_service.service.ts

Importamos Observable para detectar las llamadas de quienes se suscriban al service:

import { Observable } from 'rxjs';

También tiene que estar importado el HttpClientModule para realizar las llamadas http:

import { HttpClient } from '@angular/common/http';

### API rest

Solo hay que cambiar post/get/delet para cambiar el verbo.

En el caso de post, el body es el segundo parámetro (tiene que estar en formato json).

<any> es el tipo de objeto que va a devolver (el api siempre va a devolver un json). Se puede crear un tipo de objeto y ponerlo en vez de any.

analizarSentimiento(textoSinAnalizar: string):Observable<any>{

    var cuerpo = {'texto': textoSinAnalizar}

    var respuestaAPI = this.http.post<any>('http://127.0.0.1:5000/sentimiento', cuerpo);

    return respuestaAPI;

  }

### Usar en component

En el component.ts:

Cogemos el service de los providers:

@Component({

  selector: 'app-analisis-sentimientos',

  templateUrl: './analisis-sentimientos.component.html',

  styleUrls: ['./analisis-sentimientos.component.scss'],

  providers: [FlaskApiService]

})

Y si no funciona lo importamos:

import { FlaskApiService } from '../../services/flask-api.service';

Lo declaramos en el constructor, tanto httpClient como el service:

constructor(private http: HttpClient, private flask\_api: FlaskApiService) { }

Como es una llamada asíncrona nos tenemos que suscribir para que cuando reciba la respuesta haga algo:

analizarTexto(){

    this.flask\_api.analizarSentimiento(this.texto).subscribe((res) =>

    {

      console.log('Res ', res);

      console.log('Res',res.analisis.carita);

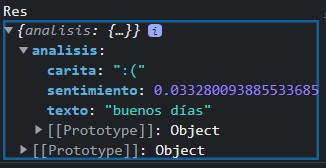
      this.texto='Sentimiento: '+res.analisis.carita+'\nResultado: '+res.analisis.sentimiento+'\nTexto: '+res.analisis.texto;

    });

  }

### Acceder al json que devuelve res

Tendremos que usar la variable res.nombreJSON.identificador.

console.log('Res',res.analisis.carita);

Si tuviera una lista se usaría [posición].

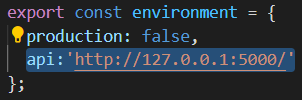
### Llamar al método desde el html

Lo hacemos con el método click y el nombre del método del .ts:

<i class="fas fa-search" (click)=" analizarTexto()"></i>

## Ruta de la API en Environment

En el fichero src/enviroment/enviroment.ts añadimos en export la ruta de la API:



En el service.ts del API importamos el enviroment:

import { environment } from '../../environments/environment';

Cuando queramos coger la ruta la cogemos del enviroment:

var respuestaAPI = this.http.post<any>(environment.api+'sentimiento', cuerpo);

# SCSS

## shared

Dentro de app creamos la carpeta shared (buenas prácticas).

Y aquí creamos un fichero .scss.

$principal: #ffffff;

$secundario: #161616;

$terciario: #d9dd01;

$acento: #b8bb15; //Destacar cosas importantes

$emisivo: #ffffff; //Links

$fugaz: #babd00; //Animaciones boton

$titulo: #161616;

$texto: #ffffff;

/\*Link a la paleta https://coolors.co/fff8f0-ff8811-f4d06f-faac40-43aa8b-897c80\*/

Buenas prácticas usar una web para elegir la paleta.

## Angular.json

Para incluir la ruta en el proyecto la añadimos en includePaths siempre desde la carpeta raíz:

  "version": 1,

  "newProjectRoot": "projects",

  "projects": {

    "Flaskcheckpoint": {

      "projectType": "application",

      "schematics": {

        "@schematics/angular:component": {

          "style": "scss"

        },

        "@schematics/angular:application": {

          "strict": true

        }

      },

      "root": "",

      "sourceRoot": "src",

      "prefix": "app",

      "architect": {

        "build": {

          "builder": "@angular-devkit/build-angular:browser",

          "options": {

            "outputPath": "dist/flaskcheckpoint",

            "index": "src/index.html",

            "main": "src/main.ts",

            "polyfills": "src/polyfills.ts",

            "tsConfig": "tsconfig.app.json",

            "inlineStyleLanguage": "scss",

            "assets": [

              "src/favicon.ico",

              "src/assets"

            ],

            "styles": [

              "src/styles.scss"

            ],

            "scripts": [],

            "stylePreprocessorOptions": {

              "includePaths": [

                "src/app/shared"

              ]

            }

          },

## Component.scss

Importamos el fichero y usamos sus variables:

@import '\_colors';

footer {

  width: 100%;

  height: 0%;

  color: $terciario;

  text-align: center;

  position: absolute;

  bottom: 0%;

}

Nota: hay páginas para transformar .css en .scss aunque css es compatible con scss pero no al revés.

# Componentes

Buenas prácticas: crear los componentes dentro de una carpeta components.

Comando

ng g c nombre-componente

Todo en minúsculas y las palabras separadas por guion medio.

## Comunicación entre HTML y TS

### Acceder al texto plano

Es **bidireccional**, es decir que podemos poner valores desde ts a html y recoger los valores del html desde el ts.

En TS declaramos la variable:

indiceLocalidad:number = 1;

En HTML usamos las {{dobles\_llaves}}:

<p>{{indiceLocalidad}}</p>

### Enviar la etiqueta HTML

No nos permite modificarla ya que recibimos una copia del estado actual de la etiqueta incluidas sus etiquetas interiores. Solo se usa a través de eventos HTML y se recibe por parámetro en TS.

En HTML usamos la almohadilla #conElNombre:

Texto

Descripción generada automáticamente

En TS usamos solamente el nombre:

Texto

Descripción generada automáticamente

### El HTML lea la variable del TS

Para hacer que el HTML lea el contenido de una variable del TS tenemos que poner el nombre del atributo entre corchetes = el nombre de la variable del TS (solo sirve si se declara como un atributo del TS):

<button [disabled]="deshabilitado"></button>

## ngModel Comunicar ts y html (solo para etiquetas imput)

Si queremos manipular el contenido del html en ejecución por ejemplo coger una entrada de texto, mostrar un resultado, etc. Necesitaremos usar la librería import { FormsModule } from '@angular/forms'; en el app.module.ts.

Primero creamos una variable de clase en el .ts del componente:

export class AnalisisSentimientosComponent  {

  textareaContent: string = '';

  json: any = false;

  imagen: any;

  result: any;

  texto = '';

Segundo colocamos la propiedad [(ngModel)] en la etiqueta a la que queramos manipular su contenido con el nombre de la variable:

<textarea class="textareaContent"  class="form-control col-md-12 textarea" type="text"

            placeholder="Inserte texto" rows="15" [(ngModel)]="texto"></textarea>

# Comunicación entre componentes (Decoradores)

Los decoradores son:

* @Input: Para recibir los datos.
* @Output: Para mandar los datos con EventEmitter.

## @Input

<https://www.youtube.com/watch?v=7Y1KpLbMTJw&t=233s&ab_channel=DevEducation>

Queremos recibir un numero para identificar cada componente generado.

### Componente que recibe el dato

En el componente que queramos recibir el dato, colocamos “Input” en los imports del .ts.

Creamos una variable y le asignamos input, por sintaxis toca inicializarla, en el caso que no se le pase el valor desde el otro componente, tomará ese valor por defecto.

import { Component, OnInit, Input } from '@angular/core';

@Component({

  selector: 'app-tarjeta',

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

  styleUrls: ['./tarjeta.component.scss']

})

export class TarjetaComponent implements OnInit {

  @Input() indiceLocalidad:number = 1;

  constructor() { }

  ngOnInit(): void {

  }

}

En el html colocamos la {{variable}} si queremos mostrarla:

<p class="card-text">Some quick example text to build on the card title id= {{indiceLocalidad}}</p>

### Componente que envía el dato

En el .ts solamente hay que tener el dato que le queramos pasar, no hay que hacer nada especial. En este caso lo hacemos desde un array:

lista:Array<any> = [0,1,2,3,4,5,6,7,8]

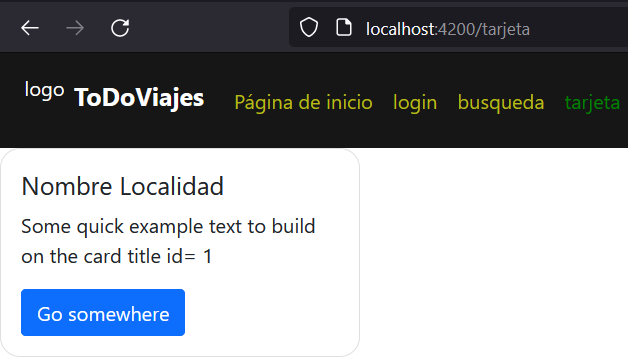
En el html hay que pasarle el dato como parámetro (con su mismo nombre) al componente destinatario:

<app-tarjeta [indiceLocalidad]="item" \*ngFor="let item of lista; index as i" class="caja2"></app-tarjeta>

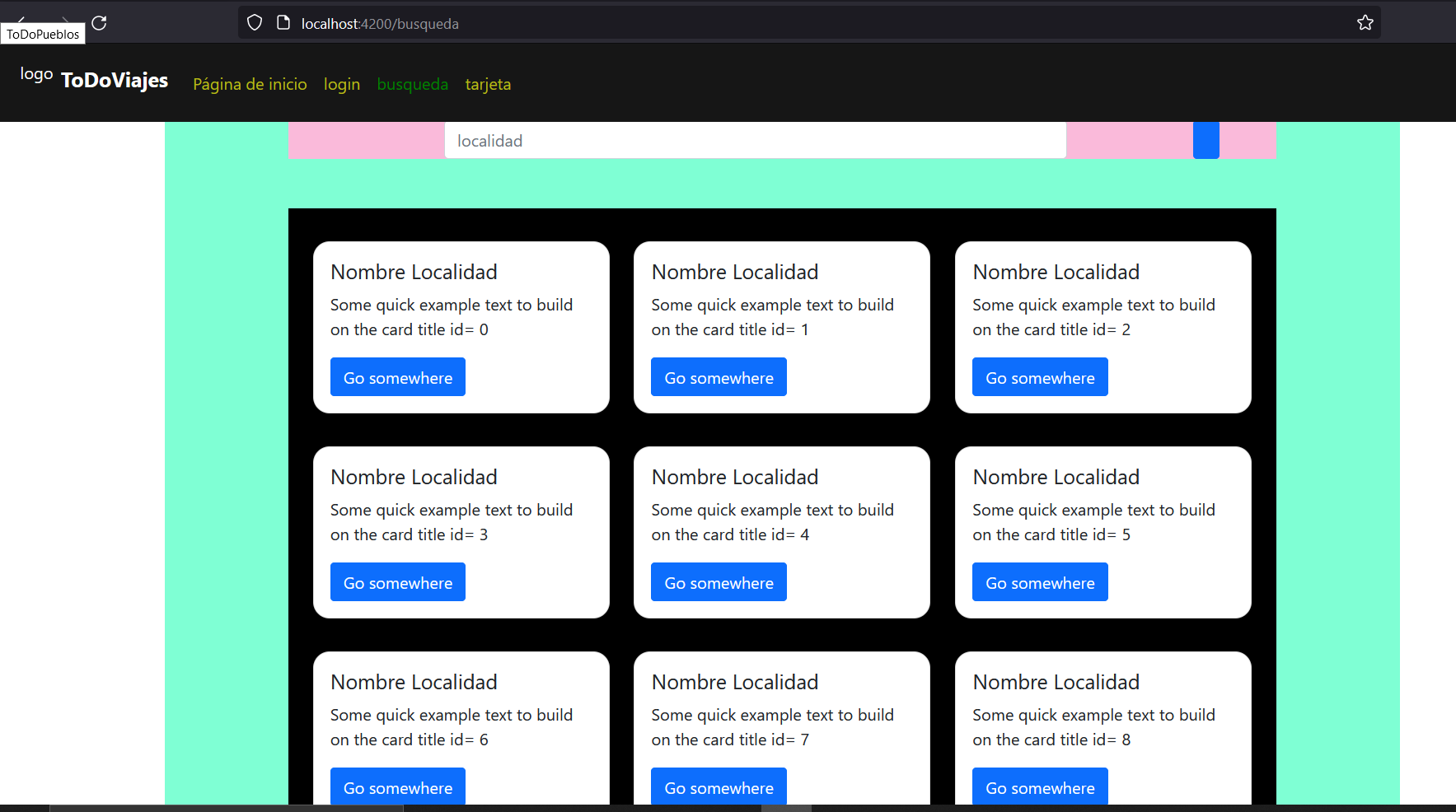
En este caso lo sacamos por medio de un ngFor.

### Resultados

Si accedemos a pelo al componente veremos que tiene el valor por defecto:



Si visualizamos la lista de los componentes vemos cómo cambia el id:



## @Output

Buscar información

# Pasar parámetros por URL

## PathParams

### App-routing.module.ts

Permitimos que la ruta pueda tener parámetros:

path:'vistalocal/:id', component: VistalocalComponent

### Componente que recibe el parámetro

#### ActivatedRoute

Importar activedRoute y ponerlo en el constructor:

import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute)

#### Coger el parámetro

Lo declaramos como atributo de la clase:

id:string = '';

En el método ngOnInit() cogemos el parámetro de la URL y se lo ponemos al atributo:

ngOnInit(): void {

    this.id = this.route.snapshot.params['id'];

  }

### Enviar el parámetro

En el componenteOrigen que va a llamar al componenteDestino le pasamos el parámetro en el href:

 <a href="vistalocal/{{indiceLocalidad}}" class="btn btn-primary">Go somewhere</a>

## QueryParams

Para leer los queryParams importamos:

import { ActivatedRoute } from '@angular/router';

Se crea un json automáticamente con los parámetros, nos subscribimos:

 this.activatedRoute.queryParams.subscribe((data=>{

        this.municipio= data["municipio"]

      }))

La URL hay que hacerla así: /?identificador1DelJson=Valor1DelJson&identificador2DelJson=Valor2DelJson

Por ejemplo:

/?municipio=Madrid&ccaa=Madrid

# Búsqueda predictiva

Referencias:

<https://youtu.be/eJw6kyjFsJY>

<https://www.npmjs.com/package/angular-ng-autocomplete>

Json ejemplo: <https://restcountries.com/>

# LocalStorage

Referencias:

<https://youtu.be/8yeDLfKUHCs>

Para leer el token:

    var tokenString:any = localStorage.getItem('token')

    var tokenJson:any = JSON.parse(tokenString)

    console.log('comprobarRoll', tokenJson.roll)

# LocalSession

Igual que localStorage pero los datos se borran al cerrar la pestaña del navegador.

# Ng-Cosas

## ngIf

Referencias:

<https://youtu.be/TDxoDRxheDs>

<li class="nav-item" \*ngIf="usuarioRegistrado()">

Si se quiere cambiar el valor sin refrescar, hay que obtener el valor por una función.

## ngFor

Referencias:

Blablablá

Importante el “:Array<any>”

busqueda:Array<any>=[]

\*ngFor="let item of busqueda; index as i"

Y para acceder a los valores usamos las llaves: {{item.foto}}

## ngClass y ngStyle

Referencias:

<https://www.youtube.com/watch?v=9ZExEHDREeQ&ab_channel=DominiCode>

NgClass aplica una clase css y ngStyle aplica las etiquetas de css directamente.

Se pueden utilizar operadores ternarios:

‘background-color’: usuario.id == 1 ? ‘si es 1’:’si no es 1’

### NgClass

Creamos la clase css:

.clor{

    background-color: aquamarine;

}

Aplicamos la clase clor si el id es igual a 2

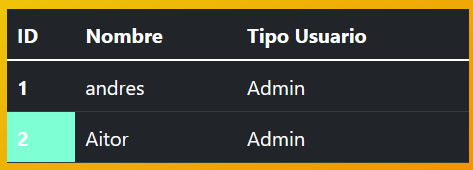
<tr \*ngFor="let usuario of datosU" (click)="editarUsuario(usuario.id)">

          <th scope="row" [ngClass]="{'clor':usuario.id == 2}">{{usuario.id}}</th>

          <td>{{usuario.nombre}}</td>

          <td>{{usuario.tipoUS}}</td>

        </tr>



### NgStyle

Ponemos en la etiqueta ngStyle las propiedades css que se van a aplicar.

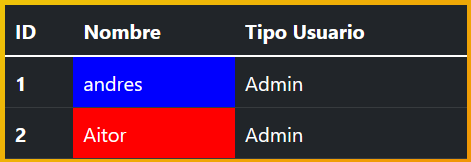
<tr \*ngFor="let usuario of datosU" (click)="editarUsuario(usuario.id)">

          <th scope="row">{{usuario.id}}</th>

          <td [ngStyle]="{'background-color': usuario.id == 1 ? 'blue':'red'}">{{usuario.nombre}}</td>

          <td>{{usuario.tipoUS}}</td>

        </tr>



# Guards Proteger URLs

Referencias:

<https://youtu.be/u6KcxQzt8lM>

En la carpeta src/app/guards. Comando ng g guard nombreDelGuard

Código:

export class RolesGuard implements CanActivate {

  constructor(private api:ApiService){}

  canActivate(

    route: ActivatedRouteSnapshot,

    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

      //para verificar

      var autorizado

      if (this.api.comprobarRoll()=='admin'){

        autorizado = true

      }else{

        autorizado = false

      }

    return autorizado;

  }

}

Se añade el service del API para poder verificar el rol desde back.

En app-routing.modules, añadimos el guard en las rutas que queramos. Se puede añadir más de un guard a una ruta.

{

  path:'admin', component: AdminComponent, canActivate:[RolesGuard]

}

# JWT autenticación

Decoder: <https://jwt.io/>

Librería para decodificar en angular: <https://www.npmjs.com/package/jwt-decode>

# Interceptor

Referencias:

<https://youtu.be/NE741Qcl3n8>

<https://youtu.be/deK_HcoPRGw>

<https://youtu.be/NE741Qcl3n8>

ng g interceptor ./interceptors/nombreInterceptor

Importamos HttpErrorResponse y catchError:

import {

  HttpRequest,

  HttpHandler,

  HttpEvent,

  HttpInterceptor,

  HttpHeaders,

  HttpErrorResponse

} from '@angular/common/http';

import { catchError } from 'rxjs/operators';

## app.module.ts

Para que el interceptor intercepte las peticiones colocamos en providers nuestro interceptor y lo importamos:

import { TokenInterceptor } from './interceptors/token.interceptor';

providers: [

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: TokenInterceptor,

      multi:true

    }

  ]

## Obtener el token

Si queremos mandar el token en el header.

## Crear headers

Creamos el json con los headers.

## Clonar request

Clonamos el request para añadirle los headers.

## Controlar el error

Controlamos el error.

# Iterfaces After

Si el HTML no se actualiza con los datos del TS hay que usar alguna de estas interfaces.

Referencias:

<https://www.youtube.com/watch?v=YUN4S56l_KY&ab_channel=MoldeoInteractive>

# Cosas para investigar

ViewChildren

QueryList