0Se sigue este curso → https://www.udemy.com/course/angular-2-fernando-herrera/learn/lecture/6397656#overview

INTRODUCCIÓN

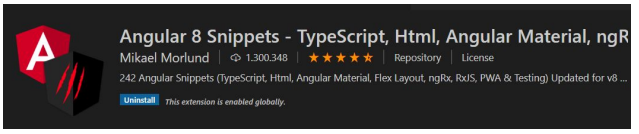
Para usar angular gastaremos las siguientes herramientas:

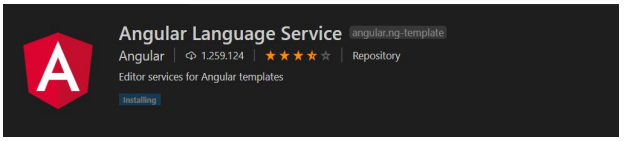
*TypeScript → JS mejorado
Lo instalaremos con el comando en CMD → `npm install -g typescript`

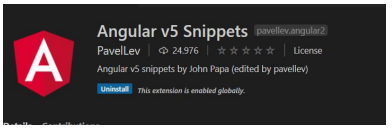
*Node JS → ?

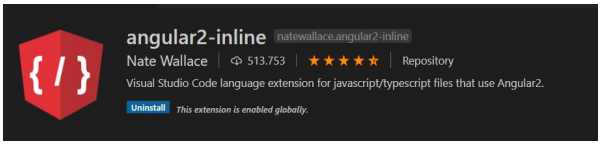
*Cliente Angular → `npm install -g @angular/cli`

Mis Snippets para Visual Studio



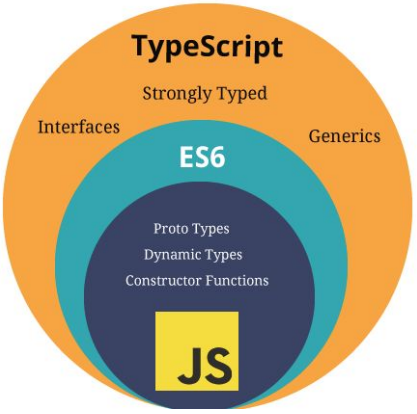






TYPESCRIPT

A resumidas cuentas **TypeScript** es JS pero bien hecho y sin tanta entropía, podemos tener el control de código que nos daba java con algunas funciones añadidas, realmente TypeScript>**EcmaScript**>JS, solo que son ñapas que se han ido metiendo para dopar el código de mayor control



Comandos para compilar TypeScript

tsc ts.ts → Para compilar un ts a js
tsc ts.ts -w → Para que el programa escuche e ir compilando auto.
tsc --init → Para crear un archivo de configuración typescript. Desde ahí se compila todo el proyecto poniendo tsc asecas.

Remember de JS:

Var → Variable global
let → Variable local, a partir de ahora usar SIEMPRE LET

-function(nombre**:string**){} → Restricciones de parámetros en funciones para que no pete el código, nos avisa el propio intellisense que está mal si la cagamos.

-let nombre**:string=** “xsoms”; → Declaras en string fijo, realmente si ponemos una variable sin especificar nada ya lo pone solo, pero bueno… no está de más ponerlo.

-let nombre**:any;** → Es como hacerlo en JS puede adoptar cualquier valor y luego redeclararlo con otro

-let nombre**:number;** → Declara número

-let texto=**`**Hola soy el número ${nombre}**`;** → Typescript da esta nueva forma de crear líneas de string, concatenado con el acento invertido y ${}, podemos meter un ENTER y será igual a un \n

-let texto= **`${**getNombre()**}`** → También podemos llamar funciones de JS desde ``

-function(variable:string=”por Defecto”){} → Se dice que va a recibir una variable y si no de mete como parámetro se pondrá por defecto el valor asignado

-function(oka**?**:string){} → valor opcional, se puede invocar a la función sin meterlo, siempre es el último valor a declarar de todos los parámetros que pongamos

Desencriptación Objeto

Teniendo el objeto

```
let a={
    atributo1=”dato”,
    atributo2=”dato”
}
```

let{ nombre, clave, poder }= a; → Va buscando por nombre y lo desencripta, de tal forma que podamos usar la variable nombre fuera sin hacer alusión a a.

Si fuéramos a desencriptar un array de string por ej.

let a:string[]= [“dato1”,”dato2”,”dato3”];

let [parametro1, parametro2, parametro3]; → De esta forma no se enlazan por variables, sino por orden secuencial, parametro1 es “dato1”

Promesas(Ejecución función asíncrona)

let promesa= new Promise(function (resolve, reject){
    console.log(“Asíncrono realizado”);
    //Ejecutamos resolve() si queremos que acabe bien
    //reject() si queremos que lanze error
})

promesa.then(function(){
    console.log("Listo");
}, function(){
    console.log("Ejecutar si sale mal");
});→ La función then se ejecuta una vez acabe el proceso, se ejecuta la primera función si sale bien usando resolve y se ejecuta la segunda función y se ejecuta el reject().

Promise() → Es una clase que ejecuta funciones en su interior, se ejecuta de manera asíncrona al resto del programa, se podría considerar un servicio

Clases Interfaces y sus importaciones

interface Objeto{
    nombre:string,
    edad:number
} → **Interfaz de TypeScript**

class Coche{
    nombre:string=”toni”;
    edad:string= undefined;

    constructor(nombre:string, equipo:string , nombreReal:string){
        this.nombre= nombre;
    } → Constructor de la clase

} → Clase en TypeScript con valores por defecto

Podemos importar las clases con import {} from “url”

```
import { Coche } from "./clases/Coche.class";
```

Y exportarlas si ponemos un export al declarar la clase

```
export class Coche{
```

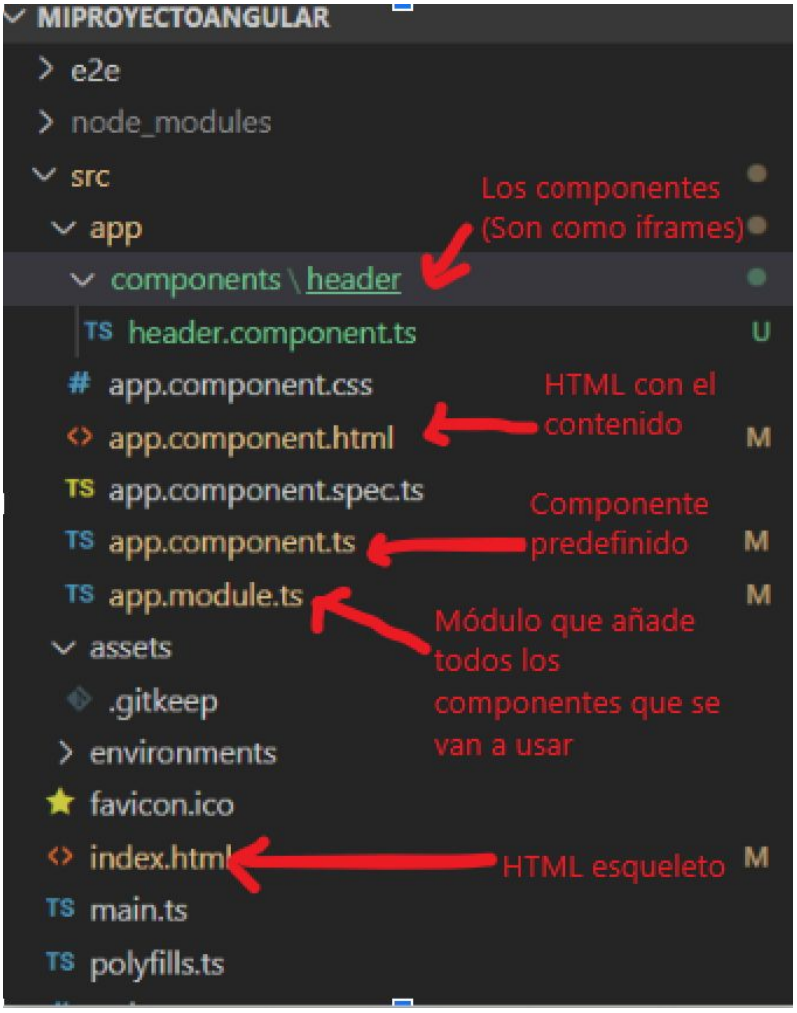
npm install
npm install bootstrap --save
npm install jquery --save
npm install popper.js --save
npm install @fortawesome/fontawesome-free

ANGULAR

ng new miProyectoAngular → Para crear el proyecto

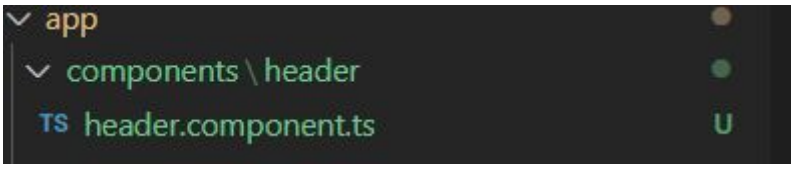
ng serve -o → Para lanzar el proyecto y abrir una pestaña nueva

Cuando creemos el proyecto se nos genera la siguiente estructura (los ficheros tocados son los más importantes)



Añadir componentes

Creamos un .component.ts como nomenclatura



En la clase:
import {Component} from '@angular/core'; → Para importar @Component

@Component{(
selector: 'app-header', → Es con lo que se invocará el componente en HTML
template: `<h1>Componente</h1>` → Lo que imprimirá el html cuando se invoque
templateUrl: './blabla.html' → Lo mismo que arriba pero sacando el html a otro lado
)}
export class HeaderComponent{} → No hace falta llenar la clase, lo importante es @Component

Lo añadimos al app.module, para avisar que hay módulo nuevo

Ya lo podemos usar en HTML como:

```
<app-header></app-header>
```

Podemos generarlo automáticamente desde la terminal con:
ng generate component components/miComponent

Por último para usar el objeto de enrutamiento ir a app.module.js e importarlo

---

<a [routerLink]="['home']" → Usamos el enrutamiento anterior para ir ahí, a /home
<li routerLinkActive="active" → Si el routerLink de arriba coincide con la página actual se añade la clase a el li, "active" es una clase de bootstrap

Si queremos hacerlo por (onclick) necesitamos usar el método backend Router

import{Router} from '@angular/router' → nos da Router, lo metemos en el constructor

this.router.navigate(['pagina', id]) → Para navegar, funciona que routerLink

```css
//Css del bueno
/* Lo de abajo no lo he hecho yo */
.animated {
    -webkit-animation-duration: 1s;
    animation-duration: 1s;
    -webkit-animation-fill-mode: both;
    animation-fill-mode: both;
  }

  .fast {
    -webkit-animation-duration: 0.4s;
    animation-duration: 0.4s;
    -webkit-animation-fill-mode: both;
    animation-fill-mode: both;
  }

  @keyframes fadeIn {
    from {
      opacity: 0;
    }

    to {
      opacity: 1;
    }
  }

  .fadeIn {
    animation-name: fadeIn;
  }
```

Condicionales y bucles

*ngIf=”variable” → Metemos esto dentro de un elemento HTML, si es true se muestra, sinó se destruye haciendo que el objeto no ocupe nada de espacio.

```
<div *ngIf="mostrar"
```

Y hemos creado un botón con un evento para cambiar la variable

```
<button (click)="mostrar= !mostrar"
```

*ngfor=”let personaje of personajes; let i = index” → Se crea un bucle for para que se creen tantos elementos como tenga el bucle, hemos usado 2 variables, una que contiene el nombre y otra que contiene el índice para operarlo si se desea, personajes es un apaño de Strings `personajes:string[]=["Toni","Ana","Jacinto"];`

```
<ul class="list-group">
   <li *ngFor="let personaje of personajes; let i = index"
        class="list-group-item">
        {{i+1}}.{{personaje}}
   </li>
</ul>
```

Instalación de módulos por ej Bootstrap

Ya sabemos que podemos inyectar Bootstrap por CDN o descargandolo manualmente, Angular nos lo gestiona para hacerlo más liviano y meterlo dentro del proyecto.

npm install bootstrap --save → Para instalar un módulo en “node_modules”

Una vez metido ir a angular.json y meter en styles los css y en scripts los js...

Enrutamiento

Angular tiene un gestor de rutas URL, para ir navegando por la misma página
Para comenzar a usar esto creamos "app.routes" dentro de app

En este fichero usamos el snippet ag-routes para que nos salga una plantilla

```
ag-routes

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

const routes: Routes = [
    { path: 'routePath', component: Component },
    { path: '**', pathMatch:'full', redirectTo: 'routePath' }
];

export const appRouting = RouterModule.forRoot(routes);
```

Importamos nuestros componentes, con ng-import

{path: "rutaURL", component: ClaseComponent} → Añadimos a Routes nuestra ruta, siendo path la ruta que usaremos en el navegador y component el component que usaremos. La última línea que nos sale es la path por defecto si no encuentra nada

export const appRouting= RouterModule.forRoot(routes, {useHash:true}); → Línea recomendada para exportar la clase de enrutamiento, el useHash true es usar # en nuestra URL por algún motivo es lo mejor

Obtener parámetros de la URL

Habiendo sido los parámetros metidos por:

```
path: 'usuario/:id',
```

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

this.activatedRoute.params.subscribe(params=>{
    this.variable= params['id'] → Cogemos el parámetro id de la URL
})

this.activatedRoute.parent.params.subscribe… → Usaremos "parent" si queremos llegar a los parámetros que tiene el padre

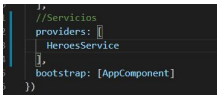
Crear servicios

Creamos la carpeta servicios en app, y dentro meteremos todos con la nomenclatura servicio.service.ts

-ng-service → Atajo para crear el servicio

Esto ya nos aporta un constructor y un espacio para crear variables, las devolvemos con getters o lo que vayamos viendo

En app.module.ts, en providers metemos el servicio



Y ya lo podemos usar en otro ts si lo importamos y se lo pasamos como parámetro por constructor

Outputs e Inputs

Nuestro fin es el poder comunicar varios html de tal forma de que podamos tener un padre y un hijo y se puedan comunicar entre sí.

INPUT

Desde el padre podemos pasar al hijo variables
<app-mierda [heroe]="heroe"> → de tal forma que le pasamos al hijo “app-mierda” heroe

Desde el hijo, importamos Input desde core:

@Input() heroe:string: → Cogemos la variable que nos ha dado el padre

OUTPUT

En el hijo importamos Output + EventEmitter:
EventEmitter es un creador de eventos que escuchará el padre

@Output heroeSeleccionado: EventEmitter<number>; → Creamos un evento que enviará un Número

this.heroeSeleccionado.emit(this.index); → Envía al padre el número del índice

En el padre:

<app-mierda (heroeSeleccionado)=”verHeroe($event)”> → Llama al método para que le

devuelva el valor, y le pasa $event para enviar el contexto

Crear Pipes

Un pipe es un editor de variable que se hacen en typescript, hay algunos ya prediseñados:

{{variable | uppercase}} → Poh eso, imprime uppercase
{{variable | lowercase}}

{{nombre | slice:0:3}} → Slice corta lo que queramos, elimina las posiciones tanto de strings como de Arrays, en el ej corta de 0-3

{{PI | number:'1.0-5'}} → Number muestra los números que queramos, 1.0 son los enteros, -5 son los decimales

{{variable | json}} → Imprime puta madre el json, consejo, usar en pre por tema espacios

{{variable | async}} → Usa promesas, este es jodido ya que la promesa tarda en llegar el valor, por lo que async dice que se espere y que no mande mierda mientras. En el back:

```
valorDePromesa= new Promise((resolve, reject)=>{
  setTimeout(()=>resolve("Llego la data!"), 3500);
});
```

Peticiones HTTP

Queremos obtener información al estilo de get y post

Metemos en el app.module.ts
import {HttpClientModule} from "@angular/common/http"; → Para hacer peticiones

Donde vayamos a hacer peticiones:

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

this.http.get("URL de consulta".subscribe(variable=>{this.variable=variable})); → Para realizar una consulta a un servidor remoto y sacar la información

Cambiar estilos desde Backend

Nota: En HTML se gasta mucho el “‘algo’”, cuando solo lleva “ ” es cuando es una variable, pero si es “‘ ’” es que es String

<p [ngStyle]=”{‘font-size’: tamany + ‘px’}”> → Usamos ngStyle para meter css desde el html pero con variable backend

<p [style.fontStyle.px]=”30”> → Otra forma mas elegante de hacerlo

<p [ngClass]=”{‘text-danger’:!propiedades.danger}” → Siendo propiedades.danger una variable del backend siendo = a true, gastamos ngClass para asignar una clase nueva dinamicamente usando propiedades del back, también las podemos quitar

Directivas personalizadas

Una directiva es una instrucción de estas <p [blabla]=”’string’”> cuando ejecutamos la directiva va al backend y se ejecuta ts

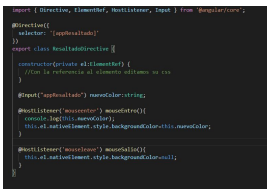
ng g d directiva → Para crear una directiva por terminal

ng-directive para generar la directiva en VS

import{Directive, ElementRef, HostListener, Input} from “@angular/core”;

Directive → Nos da la gestión de la directiva
ElementRef → Nos deja acceder a los parámetros del elemento que la ejecuta
HostListener → Usa eventos, los que queramos
Input → Cogerá el parámetro que nosotros le pasemos
this.elementRef.nativeElement.style.backgroundColor= “red” → Ejemplo de ElementRef

Se llamará a la directiva con el selector
Ejemplo perfecto:



Enrutamiento con hijos

Es como el enrutamiento normal con su app.routes.ts pero vamos a makearlos

Donde hay un archivo de rutas hay que meter en el HTML
<router-outlet></router-outlet> → Es lo que le dice al html que surque el resto de los components, se suele meter en el app.component, ahora...si creamos una ruta padre hay que meterlo ahí para que gestione sus hijos

En el Routes es igual, solo que para decir los hijos usamos children en el json

```
const routes: Routes = [
    { path: 'home', component: HomeComponent },
    {
        path: 'usuario/:id',
        component: UsuarioComponent,
        children:[
            { path: 'nuevo', component: UsuarioNuevoComponent },
            { path: 'editar', component: UsuarioEditarComponent },
            { path: 'detalle', component: UsuarioDetalleComponent },
            { path: '**', pathMatch:'full', redirectTo: 'nuevo' }

        ]
    },
    { path: '**', pathMatch:'full', redirectTo: 'home' }
];
```

Switch case con Angular

En lugar de hacer un trillon de *ng-if chupando muchos recursos/tiempo al programar podemos usar un switch case de angular.

<div [ngSwitch]=”variable”>
    <div *ngSwitchCase=”’aaaa’”> → Si variable=”aaaa” ps se mete
    <div *ngSwitchDefault> → Si no se cumple ningún case se activa