Angular 4.

INSTALACIÓN.

1. *Node.js V.x.x.x*

- Comprobar versión de node: *sudo npm –version*
- Limpiar cache de npm: *sudo npm cache clean -f*
- Instalar herramienta administración node: *sudo npm install -g -n*
- Instalar ultima versión estable: *sudo n stable*
- Instalar una versión en concreto: *sudo n 6.11.2*
- Actualizar la ultima versión de npm: *sudo npm update npm .g*
- Comprobar la versión de node instalada: *node -v.*

2. *Typescriptlang*

3. *cli.angular.io*

- Desistalar angular cli.
  - *npm uninstall -g angular-cli*
  - *npm uninstall -save-dev angular-cli*
- Actualizar el Global Package de angular cli.
  - *npm unistall -g @agular/cli*
  - *npm cache verify*
  - # if npm version is < 5 the use *npm cache clean*
  - *npm install -g @angular/cli@latest*
- Actualizar localmente en su proyecto (Package.json)
  - *npm install -save-dev @angular/cli@latest*
- Auditar vulnerabilidades: https://www.todojs.com/npm-audit-avisa-de-vulnerabilidades/

4. ionic

TIPESCRIPT y ES6

5. Conceptos:

- Los ficheros deben terminar en ‘*.ts’.
- Para compilar los ficheros y convertirlos en ‘*.js’ se utiliza ‘tsc nombre.ts’.
  - ‘tsc nombre.ts -w’ hace observable al fichero.
- Para definir el proyecto de TypeScript se utilizar ‘tsc --init’. Esto crea un fichero json ‘tsconfig.json’ donde se define las reglas para el compilador, solo hace falta introducir ‘tsc’ para compilar todos los ficheros ‘ts’ que se encuentren en la carpeta donde esta el fichero ‘tsconfig.json’.
  - “target”: “es5”. Muesta la versión de ECMAScript (javascript) que soporta el código generado, generalmente se deja 5 que es la aceptada en todos los navegadores.
  - “checkJs”: true. Permite que de errores en los archivos de Javascript.
  - “noImplicitAny”: true. Lanza un error cuando algunas de las expresiones tiene una ‘any’ como tipo.

6. Definición de variables y constantes.

- ‘let nombre_variable;’ define una variable.
- ‘const NOMBRE_CONSTANTE = “valor_constante;”’ define una constante.
- Los ficheros deben terminar en ‘*.ts’.
- Para compilar los ficheros y convertirlos en ‘*.js’ se utiliza ‘tsc nombre.ts’.
- Si definimos una variable y la inicializamos a un tipo de dato no se puede introducir otros datos de tipo diferente. ‘let nombre=”pepe”; nombre=1; //FALLO’
- Definir variable con un tipo ‘let nombre:[tipo];’, ‘let nombre:string;’.
- El tipo ‘any’ permite que la variable acepte cualquier tipo de valor.
- Crear un objeto anónimo sin tipo.

```
let spiderman = {
  nombre: ‘Peter’,
  edad: 30
}
```

  - Si al objeto le queremos modificar los parámetros no podemos añadir ni quitar parámetros porque fallará.

7. Excluir archivos a traducir.

- En el fichero tsconfig.json al principio del mismo debemos añadir ‘”exclude”: [“[nombre_directorio]”]’, que es un arreglo con todos los nombres de los directorios que no queremos que se traduzcan.

8. Plantillas literales del ES6

- Se definen entre ‘`’ y ’``’
- Para mostrar el valor de una variable se utilizar ‘${nombre_variable}’
- Se permite realizar operaciones de javascript dentro de ‘${...}’ y llamar a métodos.
- Ejemplo:

```
let nombre:string = "Fermín";
let apellido:string = "Martín";
let edad:number = 20;
let texto = `Hola,
        ${ nombre } ${ apellido } (${ edad })`;
console.log(texto);
```

- Se permite utilizar dentro de la plantilla la multilinea ‘/n’ introduciendo un ‘INTRO’ en la cadena.

9. Parámetros opcionales, obligatorios y por defecto en las funciones.

- Obligatorio: *function activar(quien:string){…}*
- Por defecto: *function activar(quien:string=”yo”){…}*
- *Opcionales:*

```
function activar(quien?:string){
    if(quien){ //Se ha definido un valor para el parámetro
    }else{//No se ha definido ningún valor para el parámetro
    }
}
```

  - Los parámetro obligatorios deben ir al principio, después los de por defecto y por ultimo los opcionales.
  - Para usar un parámetro opcional hay que rellenar los anteriores parámetros.

10. Funciones de flechas.

- *‘let miFuncionF3=(a: number, b: number) => a+b;’*

```
let nombre="Pedro";

let hulk = {
  nombre:"Hulk",
  smash(){
    /*setTimeout(
      function(){
```

```
      console.log(this.nombre + " smash!!");
    }, 1500
  );*/
  setTimeout(()=> console.log(this.nombre + " smash!!"),1500);
 }
}

hulk.smash();
```

11. Destructuración de Objeto.

- En un objeto.

```
let avenger = {
 nombre:"steve",
 clave:"Capitan America",
 poder:"Escudo"
}

//let nombre = avenger.nombre;
//let clave = avenger.clave;
//let poder = avenger.poder;

let{nombre, clave:titulo, poder} = avenger;

console.log(nombre, titulo, poder);
```

  - Si se utiliza ‘:alias’ después del nombre de la variable del objeto en destructuración se define un alias para la variable no un tipo, para definir el tipo de variable hay que indicarlo cuando se declaran dentro del objeto.

- En un método.

```
const extraer = ({nombre, poder}: any) => {
  console.log(nombre);
  consoel.log(poder);
}

extraer(avenger);
```

  - Extrae el contenido de los campos nombre y poder.

- Destructuración en array ‘arreglo’

```
let avenger:string[] = ["Thor","Steve","ironman"];
let [ heroe1,heroe2,heroe3] = avenger;
console.log(heroe1,heroe2,heroe3);
```

- Se utiliza ‘[]’ en vez de ‘{}’.
    - Se recuperan en orden, por lo que si queremos recuperar solo el tercer registro se debe definir los anteriores vacíos ‘*let [, ,heroe3] = avenger;*’, solo recupera el tercer registro.

12. Promesas en ES6

- Da la posibilidad de ejecutar una tarea o alguna función cuando una tarea asíncrona se termina.

```
let prom1 = new Promise(function(resolve, reject){
  setTimeout(()=>{
    console.log("Promesa terminada");

    //Termina bien
    resolve();

    //Termina mal
    //reject();
  }, 1500);
});

prom1.then(function(){
  console.log("Ejecutarme cuando se termina bien!");
},
function(){
  console.log("Ejecutarme cuando se termine mal!");
});
```

13. Interfaces de TypeStript

- Se declaran con la palabra reservada ‘Interface’.

```
interface Xmen{
  nombre:string,
  poder:string
}

function enviarMision(xmen:Xmen){
  console.log("Enviando a " + xmen.nombre);
}

function volverCuartel(xmen:Xmen){
  console.log(xmen.nombre + " esta volviendo al cuartel");
}

let heroe:Xmen = {
  nombre: "pepe",
  poder:"fuerza"
}

enviarMision(heroe);
volverCuartel(heroe);
```

14. Clases TypeScript

- Definición

```
class Avenger{
  nombre:string="Ant Man";
  equipo:string;
  nombreReal:string;
  puedePelear:boolean;
  peleasGanadas:number;
}

let antman:Avenger = new Avenger();

console.log(antman);
```

- Constructor, permite inicializar las propiedades de un objeto por medio de parámetros o ejecutar algún tipo de código antes que se instancie el objeto.

```
class Avenger{

  …

  constructor(nombre:string, equipo:string, nombreReal:string){
    this.nombre=nombre;
    this.equipo=equipo;
    this.nombreReal=nombreReal;
    console.log("Se ha creado un objeto Avenger");
  }
}

let antman:Avenger = new Avenger("Curro Jimenez","Trabuco","Francisco Jimenez");
antman.puedePelear=true;
console.log(antman);
```

15. Módulos.

- Permiten dividir nuestro código en varios archivos, para que sea mas fácil de mantener.
  - Se definen mediante ficheros con extensión '.class.ts', por ejemplo *'xmen.class.ts'*
  - Dentro se definen las clases, estas clases para que se puedan acceder desde el fichero donde se importa el módulo, hay que interponer antes de la definición la palabra reservada 'export', por Ejemplo *'export class Xmen{...}'*
- Importar un módulo en el fichero de TypeScript.
  - Al principio del fichero definimos la importación. 'import {nombre_clase}' from "ruta del módulo de la clase"
  - Ejemplo: *'import {Xmen} from "./clases/xmen.class";'*.
- Reducir las líneas de código para importar módulos.
  - Nos creamos un archivo por ejemplo 'index.ts' en la carpeta donde están los módulos. Dentro del ficheros definimos la exportación de estos, por ejemplo:

```
/*Al estar en la misma carpeta que los módulos no hace falta para definir en la ruta de donde se encuentra el módulo la carpeta*/
export {Xmen} from "./xmen.class";
export {Villano} from "./villano.class";
```

- Para importarlo en el fichero de TypeScript . *'import {Xmen,Villano}' from "./class/index";'*

16. Decoradores de Clases.

- Descripción
  - Es una función cualquiera.
  - para usarlo se introduce antes del nombre de la clase *'@'* seguida del nombre del decorador.
  - Los decoradores para las clases automáticamente envían como parámetro el constructor por ejemplo.

```
function consola(constructor:Function){
  console.log(constructor);
}

@consola
class Villano{
  constructor(public nombre:string){

  }
}
```

- Como los decoradores son una funcionalidad especial se puede compilar de dos formas
  - Introduciendo en la consola '*tsc –experimentalDecorators'*.
  - Configurando el archivo *'tsconfig'* añadiendo esa caracteristica.

```
"experimentalDecorators": true,
```

ANGULAR ENTORNO Y ESTRUCTURA.

17. Crear entorno local de angular.

- Versionamiento de angular. Ejemplo 7.0.2.
  - El ultimo número el 2 representa parches, retro compatible, arreglo de errores.
  - El número del medio representa una actualización menor, agrega funciones, retro compatible.
  - El primer número representa la versión de Angular. Puede ser retro compatible con la versión anterior o no.

- En la ventana de comando ejecutamos lo siguiente.
  - Ejecutamos ‘ng -v’ para ver la versión y que todo se esta instalado correctamente el servidor
  - ‘ng new my-app’ crea un nuevo proyecto angular que se llama my-app.
  - Dentro de la carpeta del proyecto que hemos creado ejecutamos el comando ‘ng server -o’ para inicializar el servidor.

18. Estructura del proyecto.

- Carpeta ‘e2e’: Se encuentran los ficheros para realizar pruebas unitarias de la aplicación.
- El fichero ‘.angular-cli.json’ o 'angular.json': El fichero de configuración del proyecto donde podemos hacer referencia a ficheros globales de css y js. Parámetros:
  - proyect.name: Nombre del proyecto
  - apps.prefix: Prefijo que se utiliza cuando se crean los componentes.
  - apps.outDir: Ruta donde se va a compilar el proyecto.
  - apps.root: Ruta principal del código.
  - apps.index: Archivo indice html de la aplicación.
  - apps.main: Archivo indice del ts.
  - apps.polyfills:Achivos de configuración para dar soporte por ejemplo a IE9, IE10, IE11.
  - apps.tsconfig: Archivo principal de configuración de typescript.
  - apps.assets; Archivos estáticos de la página web, como imágenes, iconos, etc. Estos archivos son accesibles desde todo el código.
  - apps.styles: Archivos de estilos del proyecto.
  - apps.scripts: Incluir archivos js en el proyecto.
  - apps.optimization: Optimiza la aplicación.
- El fichero ‘.editorconfig’: El fichero de configuración del editor.
- El fichero ‘.gitignore’: En este fichero se encuentran todos los archivos que se quieren ignorar para subir el proyecto a un repositorio github y compartirlo con otros usuarios.
- El fichero ‘karma.conf.js’: Este fichero es utilizado para realizar pruebas unitarias.
- El fichero ‘package.json’: El fichero sirve para reconstruir el proyecto. Incluye las dependencias del proyecto y paquetes.
  - scripts: Aparecen los script de ejecución de angular y los alias para ejecutarlos
  - dependencies: Son dependencias generales para todas las aplicaciones angular.
  - devdependencies: Son dependencias que solo son necesarias para desarrollar la aplicacion.
- El fichero ‘protractor.conf.js’: Es el fichero de configuración para realizar pruebas.
- El fichero ‘README.md’: Guarda información sobre el proyecto.
- El fichero ‘tsconfig.json’: Es el archivo de configuración de TypeScript.

- El fichero ‘tslint.json’: Ayuda a que lo errores se vean bien en el editor, auto completado del del lenguaje typescript.
- Carpeta ‘src’: Es donde se guarda todo el código de la aplicación.
  - Carpeta ‘app’: Contiene la primera pantalla que aparece cuando se abre cuando se ejecuta la aplicación de angular. Ficheros:
    - ‘app.component.ts’: Es el primer componente que estamos cargando en la aplicación de angular. Indica que hay que hacer cuando alguien utilice el ‘app-root’ de este componente en alguna parte de la página.
    - ‘app.module.ts’: Organiza los ficheros de angular.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

    - ‘declarations’: Es una lista con el nombre de todos los componentes.
    - ‘imports’: Le indicamos la configuración global de por ejemplo conexiones a la base de datos, mapas, llaves globales para conectarnos a googleMap, etc...
    - ‘providers’: Servicios que son ficheros que se van a compartir por varios componentes de la aplicación.
    - ‘bootstrap’: Indica la página que se quiere lanzar al principio de la ejecución.
  - ‘app.component.css’: Da estilo solo al componente. Si queremos dar estilo a toda la página utilizamos el ‘*src/styles.css*’
  - ‘app.component.html’: Es el código html que el componente va a mostrar.
  - ‘app.component.spec.ts’: Se utilizan para realizar pruebas sobre el componente.
- Carpeta ‘assets’: Se guarda todos los recursos estáticos como imágenes, archivos de música, videos, etc...
- El fichero ‘favicon.ico’: Es el icono se que muestra en el navegador de la aplicación.
- El fichero ‘index.html’: Es el fichero html inicial de nuestra aplicación.
  - Etiqueta <base href=”/”> indica a angular donde están los archivos, los recursos, etc.
  - Etiqueta <app-root></app-root>: Es el componente principal de la aplicación.
- El fichero ‘main.ts’: Es el que ejecuta o usa el app.modulo.

- Si esta habilitado el modulo de producción (environment.prod.ts) habilita una serie de caracteristicas que son para producción para optimizar variables, etc.

```
if (environment.production) {
  enableProdMode();
}
```

  - El fichero ‘polyfills.ts’: Nos ayuda aumentar la compatibilidad con nuestra aplicaciones.
  - El fichero ‘styles.css’: Es el archivo global de estilos de la aplicación.
  - El fichero ‘test.ts’: Contiene test de pruebas.
  - El fichero ‘typings.d.ts’: Sirve para definir variables globales a lo largo de la aplicación.
- La carpeta ‘node_modules’: Contiene todos los módulos y librerías de angular.
- La carpeta 'environments': Tenemos opciones para el entorno de producción y desarrollo.
  - environment.prod.ts:
  - environment.ts
- El fichero browserslist: Indica que navegadores soporta la aplicación, en formato humano que se pueda entender.
- El fichero tsconfig.app.json: Es el exclusivo de la aplicación no el global.

2. Utilizando Bootstrap 4.
   - La página oficial es https://v4-alpha.getbootstrap.com/ .
     - Debemos importar las siguientes librerías:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"
integrity="sha384-rwoIResjU2yc3z8GV/NPeZWAv56rSmLldC3R/AZzGRnGxQQKnKkoFVhFQhNUwEyJ"
crossorigin="anonymous">
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"
integrity="sha384-A7FZj7v+d/sdmMqp/nOQwliLvUsJfDHW+k9Omg/a/EheAdgtzNs3hpfag6Ed950n"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/tether/1.4.0/js/tether.min.js" integrity="sha384-DztdAPBWPRXSA/
3eYEEUWrWCy7G5KFbe8fFjk5JAIxUYHKkDx6Qin1DkWx51bBrb" crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/js/bootstrap.min.js" integrity="sha384-
vBWWzlZJ8ea9aCX4pEW3rVHjgjt7zpkNpZk+02D9phzyeVkE+jo0ieGizqPLForn"
crossorigin="anonymous"></script>
```

     - Estas librerías las introducimos en el fichero ‘*index.html*’ dentro de la etiqueta ‘*<head>*’
   - Instalar librerías necesarias en node_modules.
     - Otra forma es instalar las librerías para el proyecto y hacer referencia a ellas en el fichero de configuración angular-cli.json.

```
"styles": [
  "styles.css",
  "../node_modules/bootstrap/dist/css/bootstrap.min.css"
```

```
    ],
    "scripts": [
      "../node_modules/popper.js/dist/umd/popper.min.js",
      "../node_modules/jquery/dist/jquery.slim.min.js",
      "../node_modules/bootstrap/dist/js/bootstrap.min.js"
    ],
```

3. Crear un componente de Angular y Bootstrap.

   - Crear un fichero '*nombreComponente.component.ts*' en la ruta '*src/app/components/*'.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-header',
  template: `<h1>Este es el header component</h1>`
})
export class HeaderComponent {

}
```

     - Dentro de @Component hay dos propiedades:
       - '*selector*': Nombre de como queremos que se pueda hacer referencia a nuestro componente en el html.
       - '*template*': Plantilla html del componente.
     - Definimos el nombre de la clase de nuestro componente a exportar.

   - Modificamos el fichero '*app.module.ts*'.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { HeaderComponent } from './components/header.component';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

     - Creamos una importación de componente.

    - Añadimos el componente al array ‘*declarations*’ declaración de modulos.
- Hacer uso del componente en la página web usamos el selector del componente.

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

- Para introducir el código html del componente podemos crear un fichero html en la ruta *src/app/components/*’ con el mismo nombre que el componente pero con la extensión html, este fichero puedo contener el código Bootstrap de una template.
    - Para hacer referencia al fichero html del componente en el fichero header.component.ts debemos cambiar la Clave ‘template’ por ‘templateUrl’ y introducir la dirección de la página html.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-header',
  templateUrl: 'header.component.html'
})
export class HeaderComponent {

}
```

4. Crear un componente automáticamente.
    - En la consola dentro de la carpeta del proyecto introducimos los siguiente:
        - *ng g c ruta_carpetas/nombre_componente*
    - *Crea el componente sin css.*
        - *ng g c ruta_carpetas/nombre_componente -is*

5. Componentes y directivas estructurales
    - Componentes: Son pequeñas clases que cumplen una tarea específica, los componentes tienen un decorador específico.
    - Directivas estructurales: Son instrucciones que le dicen a la parte del HTML que tiene que hacer.
        - *ngIf: Se encarga de mostrar o ocultar elementos html en la página web.
            - Directiva: *ngif="expression"

              expression: tiene que ser una expresión javascript valida que devuelva un true/false.
            - componente.component.ts

```
export class BodyComponent {
  mostrar:boolean = false;
  frase:any = {
    mensaje:"Un gran poder conlleva una gran responsabilidad",
    autor:"Ben Parker"
  }
}
```

- componente.component.html

```
  <div *ngIf="mostrar" class="card card-inverse card-primary mb-3 text-center">
    <div class="card-block">
      <blockquote class="card-blockquote">
        <p>{{frase.mensaje}}</p>
        <footer>{{frase.autor}}</footer>
      </blockquote>
    </div>
  </div>
```

- *ngfor: Se encarga de hacer repeticiones de elementos html en la página.
  - Directiva: *ngFor="let item of list, let idx = index "

    list: Nombre del objeto que contiene la lista o array de elementos.

    Index: Indices de la lista o array a recorrer.

  - componente.component.ts

```
export class BodyComponent {
  personajes:String[] = ["Spiderman", "Superman", "Batman", "Venon", "Dr Bad"];
}
```

  - *componente.component.html*

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

6. Crear rutas en Angular.

- En la carpeta ‘app/components’ nos creamos un fichero llamado ‘app.routes.ts’. En este fichero definiremos las rutas de nuestra proyecto.

```
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
```

```
const APP_ROUTES: Routes = [
  { path: 'home', component: HomeComponent },
  { path: '**', pathMatch: 'full', redirectTo: 'home' }
];

/*No se usa Hash*/
/* http://localhost:4200/home */
export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);

/* Se usa Hash cuando las rutas llevan parámetro
/* http://localhost:4200/#/home */
/* export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES, {useHash:true});*/
```

- El **path:'**'** represena a la ruta por defecto, la que va a tomar si no se hace referencia a ninguna las definidas en el fichero.

- En el fichero app.modules.ts hay que importar el fichero de rutas, y en **@NgModule** a las importaciones

```
//Rutas
import { APP_ROUTING } from './app.routers';

//Servicios

//componentes
import { AppComponent } from './app.component';
import { NavbarComponent } from './components/shared/navbar/navbar.component';
import { HomeComponent } from './components/home/home.component';
import { AboutComponent } from './components/about/about.component';
import { HeroesComponent } from './components/heroes/heroes.component';

@NgModule({
  declarations: [
    AppComponent,
    NavbarComponent,
    HomeComponent,
    AboutComponent,
    HeroesComponent
  ],
  imports: [
    BrowserModule,
    APP_ROUTING
  ],
```

- Indicamos a Angular donde queremos que renderize la página

```
<div class="container">
  <router-outlet></router-outlet>
</div>
```

- Como no hemos definido ninguna ruta en el componente html de la barra de navegación en los dos enlaces nos renderiza la misma página, la de por defecto ‘home.html’

```
  <ul class="navbar-nav mr-auto mt-2 mt-lg-0">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
  </ul>
```

- Moverse entre páginas
  - Para ello utilizamos un elemento llamado ‘routerLink’.
  - Con el ‘routerLinkActive’ definimos que estilo queremos para la pestaña seleccionada.

```
  <ul class="navbar-nav mr-auto mt-2 mt-lg-0">
    <li class="nav-item" routerLinkActive="active">
      <a class="nav-link" routerLink="home">Home</a>
    </li>
    <li class="nav-item" routerLinkActive="active">
      <a class="nav-link" [routerLink]="['heroes']">Heroes</a> //La ruta lo controla angular
    </li>
    <li class="nav-item" routerLinkActive="active">
      <a class="nav-link" [routerLink]="['about']">About</a>
    </li>
  </ul>
```

7. Servicios.
   - Crear el servicio.
     - Se crean en una carpeta en el app llamada servicios con la siguiente nomenclatura ‘nombreServicio.service.ts’
     - Para definir que es un servicio utilizamos el decorador *‘@Injectable()’*.

```
import { Injectable } from '@angular/core';

private heroes:Heroe[] = [
  {
```

```
      nombre: "Aquaman",
      bio: "El poder más reconocido de Aquaman es la capacidad telepática para ...",
      aparicion: "1941-11-01",
      casa:"DC"
    },
    …..
 ];

@Injectable()
export class HeroesService {
  constructor() {
    console.log("Servicio listo para usar!!");
  }
}

getHeroes():Heroe[]{
   return this.heroes;
}

}
export interface Heroe{
  nombre:String;
  bio:String;
  img:String;
  aparicion:String;
  casa:String;
}
```

- Indicar a Angular la existencia del servicio en el fichero ‘app.module.ts’.

```
//Servicios
import { HeroesService } from './servicios/heroes.service';

//componentes
import { AppComponent } from './app.component';
import { NavbarComponent } from './components/shared/navbar/navbar.component';
import { HomeComponent } from './components/home/home.component';
import { AboutComponent } from './components/about/about.component';
import { HeroesComponent } from './components/heroes/heroes.component';

@NgModule({
  declarations: [
    AppComponent,
    NavbarComponent,
    HomeComponent,
    AboutComponent,
    HeroesComponent
  ],
  imports: [
```

```
  BrowserModule,
  APP_ROUTING
 ],
 providers: [
  HeroesService
 ],
```

- Si se define el servicio dentro del componet ese servicio esta instanciado en solo para ese componet, si despues se define en otro componet ese servicio es una instancia diferente al anterior.

```
...
@Component({
 selector: 'app-heroes',
 templateUrl: './heroes.component.html',
 providers: [HeroesService]
})
...
```

- Importar el servicio en el componente para su utilización.
  - Lo importamos con ‘import’
  - En el constructor declaramos una propiedad privada del tipo del servicio. Cuando se ejecuta el constructor se instancia el servicio quedando listo para su utilización.

```
import { Component, OnInit } from '@angular/core';
import { HeroesService, Heroe } from '../../servicios/heroes.service';

@Component({
 selector: 'app-heroes',
 templateUrl: './heroes.component.html'
})
export class HeroesComponent implements OnInit {
 heroes:Heroe[] = [];

 constructor(private _heroesService:HeroesService) {
 }

 ngOnInit() {
   this.heroes = this._heroesService.getHeroes();
 }

}
```

- Mostrar los datos del servicio en la vista.

```
<div class="card-columns">
  <div class="card" *ngFor="let heroe of heroes">
    <img class="card-img-top img-fluid" [src]="heroe.img" [alt]="heroe.nombre">
    <div class="card-block">
      <h4 class="card-title">{{heroe.nombre}}</h4>
      <p class="card-text">{{heroe.bio}}</p>
      <p class="card-text"><small class="text-muted">{{heroe.aparicion}}</small></p>
      <button type="button" name="button" class="btn btn-outline-primary btn-block">
        Ver más...
      </button>
    </div>
  </div>
</div>
```

8. Rutas con parámetros Router y ActivatedRoute

- Configuración fichero de rutas.
  - En el fichero '*app.routers.ts*' hay que importar la ruta.
    - A la ruta se le puede añadir el parámetro o parámetros que vamos a recibir.

```
import { HeroeComponent} from './components/heroe/heroe.component';

const APP_ROUTES: Routes = [
  { path: 'heroe/:id', component: HeroeComponent },
  { path: '**', pathMatch: 'full', redirectTo: 'home' }
];

export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);
```

- Utilizando RouterLink
  - En el fichero html configuramos el link de acceso a al página donde queremos acceder
    - En la ruta introducimos delante '/' para que la ruta empiece desde la página actual y no sobre <u>la raiz.</u>

```
<div class="card animated fadeIn fast" *ngFor="let heroe of heroes; let id = index">
    ...
    <a [routerLink]="['/heroe',id]" class="btn btn-outline-primary">Ver mas link...</a>
</div>
```

- Utilizando código de programación.
  - En el fichero html en el botón añadirnos un evento onclick y le pasamos el id que recuperamos del bucle anterior.

```
<div class="card animated fadeIn fast" *ngFor="let heroe of heroes; let id = index">
```

```
   ...
   <button type="button" name="button" class="btn btn-outline-primary btn-block"
      (click)="verHeroe(id)">
      Ver más...
   </button>
</div>
```

9. Recibiendo parámetros por URL ActivatedRoute.

   - En el fichero del component.ts definimos la función verHeroe que recibe como parámetro el id numérico del héroe a ver, esta función se llama desde el evento click del botón Ver más ...

```
import { Component, OnInit } from '@angular/core';
import { HeroesService, Heroe } from '../../servicios/heroes.service';
import { Router } from '@angular/router';

export class HeroesComponent implements OnInit {
….

  constructor(private _heroesService:HeroesService,
    private _router:Router) {

  }

  …

  verHeroe(idx:number){
    this._router.navigate(['/heroe',idx]);
  }

}
```

   - Importamos un ActivatedRoute y iniciamos una variable de este tipo en el constructor del component.ts que va a recibir el parámetro que enviamos en la función verHeroe.
     - El ActivatedRoute retornar un observador ‘*params.subscribe*’ que esta pendiente de todos los cambios que sucedan.

```
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { HeroesService } from '../../servicios/heroes.service';

@Component({
  selector: 'app-heroe',
  templateUrl: './heroe.component.html'
})
export class HeroeComponent {
```

```
 heroe:any = {};

 constructor(private _activatedRoute:ActivatedRoute,
             private _heroesService:HeroesService)
 {
   this._activatedRoute.params.subscribe( params =>{
     this.heroe = _heroesService.getHeroe(params['id']);
     console.log( this.heroe );
   });
 }

}
```

- En el servicio creamos una función que devuelva un héroe de array por su indice.

```
import { Injectable } from '@angular/core';

@Injectable()
export class HeroesService {

 …

 getHeroe( idx:string){
   return this.heroes[idx];
 }

}
```

10. Funcionalidad de buscar.

- En la vista de la barra de búsqueda definimos los eventos para cuando se pulsa la tecla 'Intro' en el cuadro de texto de búsqueda y click en el botón de búsqueda.
    - Con #buscarTexto definimos un nombre para el cuadro de texto el cual podemos utilizar para pasar el contenido como parámetro el valor al método buscarHeroe

```
<form class="form-inline my-2 my-lg-0">
    <input class="form-control mr-sm-2" type="text" placeholder="Buscar héroe"
       (keyup.enter) = "buscarHeroe(buscarTexto.value);" #buscarTexto>
    <button (click)="buscarHeroe(buscarTexto.value);" class="btn btn-outline-primary
        my-2 my-sm-0" type="button">Buscar</button>
</form>
```

- En el servicio de los heroes 'heroes.service.ts', se crea una función que recibe el parámetro de tipo 'string' para realizar la busqueda.
    - Se crea un array temporal el cual guarda los heroes cuyo nombre contiene el texto del parámetro de busqueda.

- Retorna el array temporal creado anteriormente.

```
buscarHeroes(termino:string){
  let heroesArr:Heroe[]=[];
  termino = termino.toLowerCase();

  for(let heroe of this.heroes){
    let nombre:string = heroe.nombre.toLowerCase();
    if(nombre.indexOf(termino)>=0){
      heroesArr.push(heroe);
    }
  }
  return heroesArr;
}
```

PIPES

1. Pipes.
   - Sirven para transformar la data de manera visual.
   - Tipos de Pipes.
     - currency
       - uppercase
       - date
     - json
     - limitTo
     - lowercase
     - async
     - decimal
     - percent
   - Ruta a documentación: https://angular.io/api?status=stable&type=pipe
2. Pipe Uppercase y Pipe: Lowercase.
   - Uppercase: Transforma toda la cadena en mayusculas

```
{{nombre | uppercase}}
```

- Lowercase: Transforma toda la cadena a minúsculas

```
{{nombre | lowercase}}
```

3. Pipe Slice
   - Elimina los x caracteres iniciales de un texto a mostrar

```
{{nombre | slice:3}}
```

   - Recupera los x primeros caracteres de un texto.

```
{{nombre | slice:0:3}}
```

   - Permite recuperar los registros de un array que se encuentran entre una posición inicial y una posición final.

```
{{arreglo | slice:1:5}}
```

   - Mostrar una lista ordenadas desde un registro en una posición inicial hasta otro registro en la posición final.

```
 <ul>
  <li *ngFor="let item of arreglo | slice:5:20">{{item}}</li>
 </ul>
```

   - Si la posición del registro final indicado es mayor al número de registros del array esto no falla y muestra todos los registros después de la posición inicial indicada.

4. Para la internacionalización en el fichero app.module.ts tenemos que realizar la siguientes configuración

```
import { BrowserModule } from '@angular/platform-browser';
import { LOCALE_ID, NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
 declarations: [
  AppComponent
 ],
 imports: [
  BrowserModule
 ],
```

```
  providers: [ {provide: LOCALE_ID, useValue: 'es'}],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5. Pipe Number(decimal): Permite dar formato a números decimales.
    - Estructura: number_expression | number[:digitInfo[:locale]]
        - digitInfo es una cadena con la que aplicamos el formato, y tiene que tener obligatoriamene esta estructura.

          {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
            - minIntegerDigits: Número mínimo de dígitos enteros.
            - MinFractionDigits: Número mínimo de dígitos decimales.
            - MaxFractionDigits: Número máximo de dígitos decimales.
        - locale es una cadena que define el uso local (LOCALE_ID)

```
  {{decimal | number:'3.2-2':'es-e'}}
```

6. Pipe Percent: Permite representar número en forma de porcentaje.
    - El funcionamiento es igual que el pipe de los number.
    - Se multiplica el numero que vamos a formatear por 100.

```
{{0.3545 | percent:'2.2-2':'es'}}
```

7. Pipe Currency: permite representar un número en forma de moneda.
    - Estructura: number_expression | currency[:currencyCode[:display[:digitInfo[:locale]]]]
        - currencyCode: Se usa utilizando los código que trae el ISO 4217
            - Al código le tenemos que indicar true para que muestre el simbolo.
        - digitInfo: Es igual al formato que utilizamos para los pipe number.

```
{{salario | currency:'EUR':true:'4.2-2':'es-ES'}}
```

8. Pipe Json: Permite formatear información de un Json, permitiendo ver el contenido del mismo en pantalla.
    - Para ver el contenido con bootstrap4 podemos meterlo en una etiqueta <pre>

```
<pre>{{heroe | json}}</pre>
```

9. Pipe Async: Permite mostrar información que viene de observables o promesas.
    - En el fichero app.component.ts definimos la promesa.

```
...
export class AppComponent {
  valorDePromesa = new Promise((resolve, reject) =>{
    setTimeout(()=>resolve('LLego la data'), 3500);
  });
}
...
```

    - En el fichero app.component.html mostramos la promesa cuando ha pasado un tiempo de retardo.

```
{{valorDePromesa | async}}
```

10. Pipe Date: Permite dar un formato a una fecha.
    - Utilizando uno de los formatos ya predefinidos en https://angular.io/api/common/DatePipe.

```
{{fecha | date:'fullDate'}}
```

    - Utilizando una versión personalizada.

```
{{fecha | date:'dd/MM/yyyy'}}
```

11. Pipes personalizados
    - En la carpeta app nos creamos otra llamada pipes donde introduciremos nuestros pipes personalizados.
    - Nos creamos un fichero con la siguiente estructura de nombre nombre_pipe_personalizado.pipe.ts
    - dentro del fichero por medio de ng2-pipe nos genera la estructura de un pipe personalizado.
        - Dentro de la etiqueta @Pipe definimos el nombre del pipe que se va ha usar desde la página html.
        - Definimos el nombre de la clase que va ha contener el pipe.
        - Dentro de la clase el método transform se ejecuta cada vez que usamos pipe en el html
            - El primer parámetro representa el valor al que se le va ha aplicar el pipe.

- Los siguientes parámetros que deseemos añadir, representan a los parámetros que queramos utilizar para realizar transformaciones al valor.
- Hay que definir el tipo del valor que vamos a retornar en la transformación. Si queremos cualquier tipo utilizamos 'any'.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'capitalizado'
})
export class CapitalizadoPipe implements PipeTransform {
  transform(value:string, todas:boolean=false): string {
    if(todas){
      return value.toUpperCase();
    }else{
      value = value.toLowerCase();
      let nombres:string[] = value.split(" ");
      for(let i in nombres){
        nombres[i] = nombres[i].charAt(0).toUpperCase()+nombres[i].substring(1);
      }
      return nombres.join(" ");
    }
  }
}
```

- El pipe hay que darlo de alta en el fichero app.module.ts

```
...
import { CapitalizadoPipe } from './pipes/capitalizado.pipe';

@NgModule({
  declarations: [
    AppComponent, CapitalizadoPipe
  ],
  ...
export class AppModule { }
```

- Utilizar el pipe en las páginas html.
  - Sin parámetros.

```
{{nombre2 | capitalizado}}
```

  - Con parámetros.

```
{{nombre2 | capitalizado:true}}
```

- Crear un pipe desde la consola.

```
Ng g p url_pipe
```

12. Pipe Domseguro: Permite crear url externas a la aplicación seguras.

- Estructura personalizada del pipe.

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';

@Pipe({
  name: 'domseguro'
})
export class DomseguroPipe implements PipeTransform {

  constructor(private domSanitizer:DomSanitizer){}

  transform(value:string, url:string): SafeResourceUrl {
    return this.domSanitizer.bypassSecurityTrustResourceUrl(url+value);
  }

}
```

- uso en la página html.

```
{{video | domseguro:'https://www.youtube.com/embed/'}}
```

13. HttpClient – Service

- Hay que importar un objeto HttpClientModule de la librería http.

```
...
import { HttpClientModule  } from '@angular/common/http';

…

@NgModule({
  declarations: [
    ...
  ],
  imports: [
    ...,
    HttpClientModule
  ],
  providers: [ ... ],
  bootstrap: [...]
})
```

```
export class AppModule { }
```

- En el servicio donde vamos a utilizar la petición http:
  - Hay que importar el objeto HttpClient de la librería http.
  - El objeto HttpClient tiene un método get que permite mediante la subcripción de un evento realizar consultas html (get, post, delete, put, etc) y recibir una respuesta del servidor.
  - Para enviar el token en los headers de la petición hay que importar un objeto HttpHeaders de la librería http.
  - Hay que crear un objeto de tipo HttpHeaders y inicializarlo con todos los heads que necesitemos.
    - Head 'authorization': Es donde se introduce el código del token.
  - Al final retornamos una colección de objetos utilizando el objeto HttpClient que es un observable que por medio del operador map nos permite realizar una consulta http y preparar o operar con los datos que nos devuelve la consulta.

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

@Injectable()
export class SpotifyService {

 artistas:any[] = [];

 constructor(public _HttpClient:HttpClient) {}

 getArtistas(){
  let url='https://api.spotify.com/v1/search?query=metallica&type=artist&limit=20';
  let headers = new HttpHeaders({
   'Authorization':'Bearer BQBB1D0PZfVFfe86ak2c7LgxrXFvk-3
                    -6h02fcY6g07VV6n2BUozhBU2-yCu7vgOkoWu9fzAKvtZyXNvspI'
  });
  return this._HttpClient.get(url,{headers:headers}).map((resp:any) => {
   this.artistas = resp.artist.items;
   return this.artistas;
  });
}
```

- Map versión de Angular 6.

```
 getQuery(query:string){
  const url = `https://api.spotify.com/v1/${query}`;
```

```
  const headers = new HttpHeaders({
    'Authorization':'Bearer ' + token
  });

  return this._HttpClient.get(url,{headers:headers});
}

getNewReleases(){
    return this.getQuery('browse/new-releases?limit=20').pipe(
      map(data => data['albums'].items)
    );
}
```

- En el componente podemos llamar al método que devuelve el observable al que podemos suscribirnos para recuperar datos de la petición.

```
...
constructor(public _spotify: SpotifyService) {
  this._spotify.getArtistas().subscribe(artistas =>{
    console.log('La respuesta ya esta lista!');
    console.log(artistas);
  });
}
...
```

- Gestión del modelo de datos 'ngModel'.
    - Importamos el objeto FormsModule de la librería forms en el app.modules.ts

```
...
import { FormsModule } from '@angular/forms';
...

@NgModule({
  declarations: [
    ...
  ],
  imports: [
    ...
    FormsModule
  ],
  providers: [ SpotifyService ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Las propiedades que añadamos a los componentes ya pueden recuperar datos de la vista y parte a formar parte del modelo utilizándolo para realizar operaciones con los datos del modelo.

```
...

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html',
  styles: []
})
export class SearchComponent{

  termino:string = '';

  constructor(public _spotify: SpotifyService) {

  }

  buscarArtista(){
    if(this.termino.length > 0){
      this._spotify.getArtistas(this.termino).subscribe(artistas =>{
        console.log('La respuesta ya esta lista!');
      });
    }
  }

}
```

- En la vista para poder tratar los datos del modelo en un componente del formulario utilizamos la etiqueta '[(ngModel)]="nombre_propiedad_modelo"' dentro de la etiqueta input.

```
  <input type="text"
         placeholder="Busque artista..."
         class="form-control"
         [(ngModel)]="termino"
         (keyup)="buscarArtista()">
```