**NODE JS**

**Proyecto REST Server**

**Cap. 100**

Iniciar nuevo proyecto

npm init -y

Crea el archivo *package.json* en el proyecto

Creamos el punto de entrada a la aplicación: Crear en la raíz el archivo *app.js*

Instalamos los siguientes componentes, express para crear un express server, y dotenv para configurar variables de entorno:

npm i express dotenv

Para chequear:

**app.js:**

console.log('hola mundo');

Luego corremos la aplicación en consola:

nodemon app

Ahora creamos nuestro Web Server, vamos a la página de *npm express:* <https://www.npmjs.com/package/express>, y copiamos la configuracíon:

const express = require('express')

const app = express()

app.get('/', function (req, res) {

res.send('Hello World')

})

app.listen(3000)

El archivo queda de esta manera

**App.js:**

const express = require('express')

const app = express()

app.get('/', function (req, res) {

  res.send('Hello World')

})

app.listen(3000)

Ya podemos abrir el navegador, vemos el *localhost* en la dirección establecida en el archivo:

<http://localhost:3000/>

Creamos en la raíz, el archivo: *.env*

Grabar adentro el nuevo puerto que vamos a utilizar:

**.env:**

PORT=8080

Hacemos el require del paquete *dotenv* para utilizar la variable de entorno y la reemplazamos

**app.js:**

require( 'dotenv' ).config();

const express = require('express')

const app = express()

app.get('/', function (req, res) {

  res.send('Hello World')

});

app.listen(process.env.PORT, () => {

    console.log( 'Servidor corriendo en puerto ', process.env.PORT );

});

INICIAR NUEVO PROYECTO

ng new gifsApp

Estrict?: y

Routing?: n

Style?: css

Descargar:

Getbootstrap.com

Codigo de instalación (jsDelivr):

<!-- CSS only -->

Abrir: src/index.html

Pegar el codigo de instalación Bootstrap dentro del <head>:

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

Iniciar el servidor:

ng serve -o

Para generar un nuevo módulo (se genera archivo.module.ts en app/nombreModulo)

ng g m dbz

Para generar un nuevo component

ng generate component <nombre>

ng g c heroes/listado

Nuevo componente sin el archivo de pruebas y la hoja de estilos:

ng g c dbz/mainPage --skip-tests -is

Para crear nuevo servicio, dentro de gifs creamos la carpeta services y dentro el archivo gifs

Ng g s gifs/services/gifs --skip-tests

Este servicio se define en el ‘root’ lo eleva en un nivel global para que esté disponible en toda la aplicación, evita tener que especificar en los providers

**Usando la API GIPHY**

Descargar: <https://developers.giphy.com/>

Daschboard

Create an app

API

Your App Name: Curso de angular de cero a experto

App Description: API para el cuso de Udemy de Angular

Copiar el link api key

Luego entrar en Docs

Buscar gif & sticker Endopoints

Luego Search Endpoint

Importar nuevo módulo en app.module.ts para que sea de manera global

import { HttpClientModule } from '@angular/common/http'

Agregar la importación también en el apartado imports:

imports: [

    BrowserModule,

    HttpClientModule,

    SharedModule,

    GifsModule

  ],

Creamos el constructor e inyectamos el servicio en gifs.services.ts:

constructor(private http: HttpClient) {}

Importar nuevo módulo en heroes.module.ts el módulo es:

FormsModule

Two way data binding: Emite como recibe eventos o propiedades

<input

                type="text"

                placeholder="Nombre"

                name="nombre"

                [(ngModel)]="nuevo.nombre"

/>

Crear nuevo componente:

ng g c dbz/personajes --skipTests

Borrar el archivo css

Eliminar la importación del css en nuevoArchivo.component.ts

Eliminar el implements onInit de la clase PersonajesComponent, junto con el constructor y la llamada

**Crear la carpeta services**

Crear archivo dbz.service.ts

Actualizar el archivo dbz.modules.ts, se agrega en un nuevo apartado:

providers: [

    DbzService

]

**PARA TIPADO**

Abrir: quicktype.io

Copiar la respuesta de Postman (en Json) y pegar en la ventana izquierda de quicktype.

En options/languaje Cambiar el lenguaje a typescript

Luego copiar el codigo (copy code)

Crear una interfase en el proyecto

Gifs/interface/gifs.interface.ts

Pegar en el archivo el codigo copiado

Cambiar el tipado any por el nuevo SearchGifResponse

**LOCAL STORAGE**

**PARA GENERAR VERSION DE PRODUCCION**

$ ng build --prod

**Cap. 121:**

Regiones del res countries.

Copiamos el end point correspondiente

Copiamos las regiones y creamos un arreglo con los elementos:

 Africa, Americas, Asia, Europe, Oceania

**por-region.component.ts:**

export class PorRegionComponent {

regiones: string[] = [ 'africa', 'americas', 'asia', 'europe', 'oceania'];

Implementamos el arreglo en nuestro html.

**por-region.component.html:**

<h3>Buscar por región <small> region activa </small></h3>

<hr>

<h5>Seleccione la región</h5>

<div class="row">

    <div class="col">

        <button \*ngFor="let region of regiones"

            class="btn btn-outline-primary">

            {{ region }}

        </button>

    </div>

</div>

Los botones que aparecen podemos dar formato en la sección stiyles de su componente, de esta manera solo afecta a los botones de este componente.

**por-region.component.ts:**

@Component({

  selector: 'app-por-region',

  templateUrl: './por-region.component.html',

  styles: [ `

    button {

      margin-right: 5px;

    }

  `

  ]

})

Para ver cual región se hizo clic para activar el botón, creamos el método correspondiente, mas otro método para dejar activado el botón.

**por-region..ts:**

regiones: string[] = [ 'africa', 'americas', 'asia', 'europe', 'oceania'];

  regionActiva: string = '';

  constructor() { }

  getClaseCSS( region: string ): string {

    return (region === this.regionActiva)

      ? 'btn btn-primary'

      : 'btn btn-outline-primary';

  }

  activarRegion( region: string ) {

    this.regionActiva = region;

  }

Implementamos los métodos, actualizamos también la variable *regionActiva*

**por-region..html:**

<h3>Buscar por región <small> {{ regionActiva | titlecase }} </small></h3>

<hr>

<h5>Seleccione la región</h5>

<div class="row">

    <div class="col">

        <button \*ngFor="let region of regiones"

            [class]="getClaseCSS( region )"

            (click)="activarRegion( region )">

            {{ region | titlecase }}

        </button>

    </div>

</div>

**Cap. 124:**

Creamos el servicio para la búsqueda con el path correspondiente

https://restcountries.com/v3.1/region/{region}

**pais-service.ts:**

buscarRegion(region: string): Observable<Country[]> {

    const url = `${ this.apiUrl }/region/${ region }`;

    return this.http.get<Country[]>(url);

  }

Inyectamos el servicio en el constructor e implementamos el método para la búsqueda por región.

**por-region..ts:**

regionActiva: string = '';

  paises: Country[] = [];

  constructor(private paisService: PaisService) { }

  getClaseCSS( region: string ): string {

    return (region === this.regionActiva)

      ? 'btn btn-primary'

      : 'btn btn-outline-primary';

  }

  activarRegion( region: string ) {

    if ( region === this.regionActiva ) { return; }

    this.regionActiva = region;

    this.paises = [];

    this.paisService.buscarRegion( region )

      .subscribe( paises => {

        this.paises = paises

      });

  }

**por-region..html:**

<div \*ngIf="paises.length > 0" class="row">

    <div class="col">

        <app-pais-tabla [paises]="paises">

        </app-pais-tabla>

    </div>

</div>

**Cap. 128:**

Creamos la lista de sugerencias, primero trabajos el html

**por-pais..html:** debajo del *div* de búsqueda y antes del <hr> y \*ngIf = “Hay error” hacemos el esquema

<ul class="list-group">

    <li class="list-group-item list-group-item-action">

        nombre pais

    </li>

</ul>

Luego personalizamos el cursor en la sección *styles* de su componente.

**En por-pais..ts**

selector: 'app-por-pais',

  templateUrl: './por-pais.component.html',

  styles: [

    `

      li {

        cursor: pointer;

      }

    `

  ]

Luego en el mismo archivo

sugerencias(termino: string): void {

    this.hayError = false;

    this.paisService.buscarPais( termino )

      .subscribe( paises => this.paisesSugeridos = paises.splice(0,5),

      (err) => this.paisesSugeridos = []

    );

}

Se debe crear también dentro de la clase, la variable tipo array:

 hayError: boolean = false;

 paises: Country[] = [];

 paisesSugeridos: Country[] = [];

En **por-pais..html:**

<ul class="list-group">

    <li \*ngFor="let pais of paisesSugeridos"

        class="list-group-item list-group-item-action">

        <a class="nav-link"

            [routerLink]="['/pais', pais.cca2]">

            {{ pais.name.common }}

        </a>

    </li>

</ul>

Para el *footer*, agregamos en

**style.css:**

body {

    position: relative;

    padding-bottom: 3em;

    min-height: 96vh;

}

footer {

    background-color: lightcoral;

    padding: 1em 0;

    text-align: center;

    position: absolute;

    bottom: 0;

    width: 100%;

}

**PIPES**

**Cap. 133:**

Crear nuevo proyecto:

$ng new 04-pipesApp

Consultar la documentación oficial de Angular en la parte de Pipes:

<https://angular.io/api/common/CommonModule#pipes>

Consultar también PRIMENG:

<https://www.primefaces.org/primeng/#/>

Ir a *Get Started* para consultas

Una vez terminada la generación del proyecto, iniciar:

$ng serve -o

Luego instalamos los nuevos componentes:

$npm install primeng primeicons

Detener el servidor para mas instalaciones:

$Ctrl + C

En la parte de *styles* de la página PrimeNG encontramos los 3 comandos a instalar:

node\_modules/primeicons/primeicons.css

node\_modules/primeng/resources/themes/lara-light-blue/theme.css

node\_modules/primeng/resources/primeng.min.css

Instalar en el archivo **angular.json** del proyecto en el apartado de *styles*:

Agregar las “” y las , respectivas para que se mantenga como un archivo JSON válido.

],

"styles": [

    "src/styles.css",

    "node\_modules/primeicons/primeicons.css",

    "node\_modules/primeng/resources/themes/lara-light-blue/theme.css",

    "node\_modules/primeng/resources/primeng.min.css"

],

"scripts": []

Luego volver a levantar la aplicación:

$ng serve

Más debajo de la página **PRIMENG** en la sección *Free Themes* podemos buscar en órden alfabetico el tema que vamos a utilizar:

primeng/resources/themes/vela-blue/theme.css

Copiamos en el archivo **angular.json** y corregimos la ruta correcta, borramos el otro tema que no utilizaremos:

"styles": [

"src/styles.css",

"node\_modules/primeicons/primeicons.css",

"node\_modules/primeng/resources/themes/lara-light-blue/theme.css", //ELIMINAR

"node\_modules/primeng/resources/themes/vela-blue/theme.css",

"node\_modules/primeng/resources/primeng.min.css"

],

**SECCION 12**

**HEROES APP – LAZY LOUD**

Iniciar proyecto: *heoresApp*

Instalamos *Angular Material*:

<https://material.angular.io/>

Entramos a: *Guides*

*Getting Started*

Obtenemos el comando de instalación:

ng add @angular/material

Seleccionar tema: Purple/Green

Set up global Angular Material Typography styles?: Y

Set up browser animations for Angular Material?: Y

Levantar la app.

Crear los siguientes MÓDULOS todos en la raiz de la app:

Auth ( autenticación )

heroes

material

Creamos una carpeta dentro de AUTHllamada PAGES

Adentro creamos los siguientes *components:*

auth/pages/login --skip -tests -is

auth/pages/registro --skip -tests -is

Creamos una carpeta dentro de HERORES*,* llamada PAGES

Adentro craemos los siguientes components:

heroes/pages/agregar --skip -tests -is

heroes/pages/buscar --skip -tests -is

heroes/pages/heroe --skip -tests -is

heroes/pages/home --skip -tests -is

heroes/pages/listado --skip -tests -is

Dentro de los module.ts NO SE EXPORTA NADA, así se carga de manera Lazy Laud

Creamos una carpeta en la raiz de la App, llamada SHARED, adetro creamos un component:

shared/errorPage --skip -tests -is

**Cap. 175:**

Creamos el módulo para nuestras rutas principales:

$ng g m appRouting --flat (flat es para que NO lo cree dentro de una nueva carpeta, sino en la raiz)

**app-routing.module.ts:**

Borrar la importación del { CommonModule }

Borrar también las declaraciones: declarations: [],

Crear la constante para las *Routes:*

*const routes: Routes = []* (import { Routes }… )

Luego crear los paths

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

import { ErrorPageComponent } from './shared/error-page/error-page.component';

const routes: Routes = [

  {

    path: '404',

    component: ErrorPageComponent // se encuentra definido de forma global en

// app.module. Para llamar a otros componentes

// deberiamos importarlos en el app.module

  },

  {

    path: '\*\*',

    component: ErrorPageComponent

  }

];

@NgModule({

  imports: [

    RouterModule.forRoot( routes )

  ],

  exports: [

    RouterModule

  ]

})

export class AppRoutingModule { }

Agregamos también el nuevo módulo creado en app.module.ts

**app.module.ts:**

 imports: [

    BrowserModule,

    BrowserAnimationsModule,

    AppRoutingModule

  ],

En

**app.component.html:** Agregamos el módulo <router-outlet> con esto le indicamos a Angular que dependiendo de la ruta en donde se encuentre, que haga match con lo definido dentro de este módulo.

<h1>Hola mundo</h1>

<router-outlet></router-outlet>

**Cap. 176:**

Creamos el módulo:

$ng g m auth/authRouting --flat

Luego en

**auth-authRouting.module.ts:**

const routes: Routes = [

  {

    path: '',

    children: [

      {

        path: 'login',

        component: LoginComponent

      },

      {

        path: 'registro',

        component: RegistroComponent

      },

      {

        path: '\*\*',

        redirectTo: 'login'

      }

    ]

  }

]

Luego mas abajo, borramos declarations: [ ] y CommonModule, agregamos los imports y exports:

@NgModule({

  declarations: [], // borrar

  imports: [

    CommonModule,

    RouterModule.forChild( routes )

  ],

  exports: [

    RouterModule

  ]

})

export class AuthRoutingModule { }

En **auth.module.ts:** Importamos el auth-routing para poder utilizarlo en ese módulo

@NgModule({

  declarations: [

    LoginComponent,

    RegistroComponent

  ],

  imports: [

    CommonModule,

    AuthRoutingModule

  ]

})

**app-routing.module.ts:**

Agregamos un nuevo path y definimos la ruta hija.

const routes: Routes = [

  {

    path: 'auth',

    loadChildren: () => import( './auth/auth.module' ).then( m => m.AuthModule )

  },

  {

    path: '404',

    component: ErrorPageComponent

  },

En **error-page.component.ts:**

Para visualizar mejor como funciona, agregamos links

<p>error-page works!</p>

<a routerLink="/auth/login">

    Ir a login

</a>

<a routerLink="/auth/registro">

    Ir a registro

</a>

**Cap. 178:**

Creamos el módulo:

$ ng g m heores/heroesRouting --flat

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

import { ListadoComponent } from './pages/listado/listado.component';

import { AgregarComponent } from './pages/agregar/agregar.component';

import { BuscarComponent } from './pages/buscar/buscar.component';

import { HeroeComponent } from './pages/heroe/heroe.component';

const routes: Routes = [

  {

    path: '',

    children: [

      {

        path: 'listado',

        component: ListadoComponent

      },

      {

        path: 'agregar',

        component: AgregarComponent

      },

      {

        path: 'editar/:id',

        component: AgregarComponent

      },

      {

        path: 'buscar',

        component: BuscarComponent

      },

      {

        path: ':id',

        component: HeroeComponent

      },

      {

        path: '\*\*',

        redirectTo: 'listado'

      }

    ]

  }

];

@NgModule({

  imports: [

    RouterModule.forChild( routes )

  ],

  exports: [

    RouterModule

  ]

})

export class HeroesRoutingModule { }

**heroes.module.ts:**

imports: [

    CommonModule,

    FlexLayoutModule,

    MaterialModule,

    HeroesRoutingModule

  ]

**app.routing.module.ts:**

const routes: Routes = [

  {

    path: 'auth',

    loadChildren: () => import( './auth/auth.module' ).then( m => m.AuthModule )

  },

  {

    path: 'heroes',

    loadChildren: () => import( './heroes/heroes.module' ).then( m => m.HeroesModule )

  },

**home.component.ts:**

Agregamos para visualizar:

<h1>Home component</h1>

<hr>

<router-outlet></router-outlet>

**heroes.routing.module.ts:**

const routes: Routes = [

  {

    path: '',

    component: HomeComponent,

    children: [

      {

        path: 'listado',

        component: ListadoComponent

      },

**Cap. 184:**

Bajamos la aplicación.

**material.module.ts:**

Borramos lo que no se va utilizar

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

import { CommonModule } from '@angular/common';

@NgModule({

  declarations: [],

  imports: [

    CommonModule

  ]

})

export class MaterialModule { }

Solo utilizaremos el *exports* para exportar todos los módulos de Angular que vamos a utilizar

Luego de la página *npm* instalamos el *Angular Flex Layout.*

<https://www.npmjs.com/package/@angular/flex-layout>

npm i @angular/flex-layout

Luego nos vamos al repositorio en Git Hab de este mismo componente:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Vamos hasta Getting Started y compiamos el link para instalar:

npm i -s @angular/flex-layout @angular/cdk

Luego relizar la importación solo en el módulo donde vamos a utilizarlo, en este caso solo en Heroes:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Heroes.module.ts:**

import { FlexLayoutModule } from '@angular/flex-layout';

import { HeroesRoutingModule } from './heroes-routing.module';

@NgModule({

  declarations: [

    AgregarComponent,

    BuscarComponent,

    HeroeComponent,

    HomeComponent,

    ListadoComponent

  ],

  imports: [

    CommonModule,

    FlexLayoutModule,

    HeroesRoutingModule

  ]

})

export class HeroesModule { }

**Cap. 185:**

Vamos a *Angular Component* y descargarmos:

**Sidenav**

Entramos en la pestaña API copiamos el enlace:

import {MatSidenavModule} from '@angular/material/sidenav';

**material.module.ts:**

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

import { MatSidenavModule } from '@angular/material/sidenav';

@NgModule({

    exports: [

        MatSidenavModule

    ]

})

export class MaterialModule { }

**heroes.module.ts:** Lo importamos aquí

imports: [

    CommonModule,

    FlexLayoutModule,

    MaterialModule,

    HeroesRoutingModule

  ]

Vamos a *Angular Component* y descargarmos:

**Toolbar**

Entramos en la pestaña API copiamos el enlace, lo pegamos en *material*:

import {MatToolbarModule} from '@angular/material/toolbar';

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

import { MatSidenavModule } from '@angular/material/sidenav';

import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({

    exports: [

        MatSidenavModule,

        MatToolbarModule

    ]

})

export class MaterialModule { }

**app.component.ts:**

Borramos el Hola mundo, solo queda el *<router-outlet>*

**home.component.ts:**

Borramos todo el contenido existente

Componente de *Angular Material:*

**Button**

Siempre en la pestaña API

import {MatButtonModule} from '@angular/material/button';

**material.module.ts:**

import { MatButtonModule } from '@angular/material/button';

import { MatSidenavModule } from '@angular/material/sidenav';

import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({

    exports: [

        MatButtonModule,

        MatSidenavModule,

        MatToolbarModule

Componente de *Angular Material:*

**Icon**

Siempre en la pestaña API

import {MatIconModule} from '@angular/material/icon';

**material.module.ts:**

import { MatButtonModule } from '@angular/material/button';

import { MatIconModule } from '@angular/material/icon';

import { MatSidenavModule } from '@angular/material/sidenav';

import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({

    exports: [

        MatButtonModule,

        MatIconModule,

        MatSidenavModule,

        MatToolbarModule

**Cap. 186:**

Componente de *Angular Material:*

**list**

Siempre en la pestaña API

import {MatListModule} from '@angular/material/list';

import { MatButtonModule } from '@angular/material/button';

import { MatIconModule } from '@angular/material/icon';

import { MatListModule } from '@angular/material/list';

import { MatSidenavModule } from '@angular/material/sidenav';

import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({

    exports: [

        MatButtonModule,

        MatIconModule,

        MatListModule,

        MatSidenavModule,

        MatToolbarModule

**home.componen.html:**

<mat-sidenav-container fullscreen>

        <mat-sidenav #sidenav mode="push">

            <!--

            <h1>Sidenav en esto se basa el ancho del material</h1>

            -->

            <mat-toolbar color="primary">

                <span>Menú</span>

                <button mat-icon-button

                    (click)="sidenav.toggle()">

                    <mat-icon>menu</mat-icon>

                </button>

            </mat-toolbar>

            <mat-nav-list>

                <a routerLink="./listado" mat-list-item (click)="sidenav.toggle()">

                    <mat-icon mat-list-icon>label</mat-icon>

                    <span>Listado de héroes</span>

                </a>

            </mat-nav-list>

            <mat-nav-list>

                <a routerLink="./agregar" mat-list-item (click)="sidenav.toggle()">

                    <mat-icon mat-list-icon>add</mat-icon>

                    <span>Añadir héroe</span>

                </a>

            </mat-nav-list>

            <mat-nav-list>

                <a routerLink="./buscar" mat-list-item (click)="sidenav.toggle()">

                    <mat-icon mat-list-icon>search</mat-icon>

                    <span>Buscar héroe</span>

                </a>

            </mat-nav-list>

        </mat-sidenav>

        <mat-toolbar>

            <button mat-icon-button

                (click)="sidenav.toggle()">

                <mat-icon>

                    menu

                </mat-icon>

            </button>

        </mat-toolbar>

        <div class="container">

            <router-outlet></router-outlet>

        </div>

    </mat-sidenav-container>

Formateamos la clase *container* en su archivo componente.

**home.component.ts:**

@Component({

  selector: 'app-home',

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

  styles: [`

    .container {

      margin: 10px;

    }

  `]

})

Creamos un nuevo botón, el *Logout.*

Luego creamos una clase global para configuar el *flex* con la clase *spacer*, esto toma todo el espacio restante y coloca el elemento al final.

**styles.css**

.spacer {

    flex: 1 1 auto;

}

Para aplicar esta clase, lo colocamos en un <span> antes del nuevo botón creado

**home.component.ts:**

  <mat-toolbar>

            <button mat-icon-button

                (click)="sidenav.toggle()">

                <mat-icon>

                    menu

                </mat-icon>

            </button>

            <span class="spacer"></span>

            <button mat-button>

                Logout

            </button>

   </mat-toolbar>

También agregamos la misma clase creada, mas arriba en el botón del menú

 <mat-toolbar color="primary">

                <span>Menú</span>

                <span class="spacer"></span>

                <button mat-icon-button

                    (click)="sidenav.toggle()">

                    <mat-icon>menu</mat-icon>

                </button>

 </mat-toolbar>

CREACION DE UN JSON SERVER

**Cap. 187:**

Instalamos desde el NPM:

<https://www.npmjs.com/package/json-server>

Instalamos de manera global:

npm install -g json-server

Descargar el archivo existente en el curso, *db.json*, crear en una carpeta llamda *05-heroes-server*

Luego, debemos levantar el servidor desde la carpeta que acabamos de crear

json-server --watch db.json

Descargar también las imágenes del curso (carpeta imágenes + archivo no-image.png), guardar en la carpeta del proyecto:

app/src/assets

**Cap. 188:**

Creamos una nueva carpeta *services* y un nuevo servicio para nuestros heroes, en: app/heroes/services

$ng g s heroes/services/heroes --stip-tests

Con esto el servicio se ejecuta de manera global en la app, se provee en el root. Si queremos ejecutar solo en el módulo, debemos proveer en el módulo *heroes.module*.

Instalamos el servicio HTTP

**app.module.ts:**

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

@NgModule({

  declarations: [

    AppComponent,

    ErrorPageComponent

  ],

  imports: [

    BrowserModule,

    BrowserAnimationsModule,

    HttpClientModule,

    AppRoutingModule

  ],

Luego ya podemos usar este servicio en el nuevo service de heroes creado, entonces lo configuarmos

**heroes.service.ts:**

export class HeroesService {

  constructor( private http: HttpClient ) { }

  getHeroes() {

    return this.http.get('http://localhost:3000/heroes')

  }

}

Inyectamos este servicio en listado.component.

**listado.component.ts:**

export class ListadoComponent implements OnInit {

  constructor( private heroesService: HeroesService ) { }

  ngOnInit(): void {

    this.heroesService.getHeroes()

      .subscribe( resp => console.log( resp ) );

  }

}

Utilzamos el <https://app.quicktype.io/> para formatear nuestros datos

Obtenemos los datos de nuestra petición GET del Postman y lo pegamos en Quicktype

Copiamos el formato obtenido, le ponemos el nombre: Heroe

Crear carpeta *interfaces* pero la utilizaremos solo dentro de la carpeta *héroes*, adentro creamos nuestro archivo para interface:

heroes/interfaces/heroe.interface.ts

**heroe.interface.ts:**

hacemos las siguientes modificaciones:

* al id le definimos como opcional: ?
* agregamos el campo alt\_img?: string. Y también lo definimos como opcional

export interface Heroe {

    id?:               string;

    superhero:        string;

    publisher:        Publisher;

    alter\_ego:        string;

    first\_appearance: string;

    characters:       string;

    alt\_img?:         string;

}

export enum Publisher {

    DCComics = "DC Comics",

    MarvelComics = "Marvel Comics",

}

Entonces modificamos el archivo correspondiente, agregando el nuevo tipo de dato creado en la interface.

**heroes.service.ts:**

import { Observable } from 'rxjs';

import { Heroe } from '../interfaces/heroe.interface';

@Injectable({

  providedIn: 'root'

})

export class HeroesService {

  constructor( private http: HttpClient ) { }

  getHeroes(): Observable<Heroe[]> {

    return this.http.get<Heroe[]>('http://localhost:3000/heroes');

  }

}

**listado.component.ts:**

export class ListadoComponent implements OnInit {

  heroes: Heroe[] = [];

  constructor( private heroesService: HeroesService ) { }

  ngOnInit(): void {

    this.heroesService.getHeroes()

      //.subscribe( resp => console.log( resp ) );

      .subscribe( resp => {

        this.heroes = resp;

      });

}

**listado.component.html:**

<ul>

    <li \*ngFor="let heroe of heroes"

    >{{heroe.superhero}}</li>

</ul>

**Cap. 190:**

Componente de *Angular Material:*

**Card**

Siempre en la pestaña API

import {MatCardModule} from '@angular/material/card';

El archivo queda de esta manera

**listado.component.html:**

<!--

<ul>

    <li \*ngFor="let heroe of heroes"

    >{{heroe.superhero}}</li>

</ul>

-->

<h1>Listado de Héroes</h1>

<mat-divider></mat-divider>

<br>

<div fxLayout="row wrap"

    fxLayout.xs="column" //.xs (si es muy pequeña) utilizo forma de columna

    fxLayoutAlign="center" // alineación de las columnas

    fxLayoutGap="20px"> // separación entre columnas 20 px.

    <div fxFlex="20" // ocupará el 20% de la columna

        fxFlex.lg="15"

        fxFlex.sm="30"

        \*ngFor="let heroe of heroes">           <!-- le indicamos que ocupará 20 px. de la columna -->

        <mat-card>

            <mat-card-header>

                <mat-card-title>{{ heroe.superhero }}</mat-card-title>

                <mat-card-subtitle>{{ heroe.alter\_ego }}</mat-card-subtitle>

            </mat-card-header>

            <img mat-card-image src="assets/heroes/{{ heroe.id }}.jpg">

            <mat-card-content>

                <h3>{{ heroe.publisher }}</h3>

                <p>

                    <strong>Primera aparición: </strong> {{ heroe.first\_appearance }}

                    <br>

                    <strong>Personajes: </strong>{{ heroe.characters }}

                </p>

            </mat-card-content>

            <mat-card-actions>          <!-- espacio reservado para botones -->

                <button mat-button color="warn">

                    Leer mas...

                </button>

                <button mat-button color="info">

                    Editar

                </button>

            </mat-card-actions>

        </mat-card>

    </div>

</div>

**Cap. 192:**

Crear la carpeta components, y adentro el componente:

$ng g c heroes/components/heroeTarjeta --stip-tests -is

Podemos eliminar todo lo que no usaremos, y agregamos el @Input para pasarle la variable

**heroe-tarjeta.component.ts:**

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

import { Heroe } from '../../interfaces/heroe.interface';

@Component({

  selector: 'app-heroe-tarjeta',

  templateUrl: './heroe-tarjeta.component.html'

})

export class HeroeTarjetaComponent {

  @Input() heroe!: Heroe;

}

Movemos de *listado.component.html* todo el código de la etiqueta <mat-card> y lo pegamos en el nuevo componente creado *(heroeTarjeta)*, en lugar del código movido dejamos el marcador <app-heroe-tarjeta> y agregamos la variable pasada con el Input.

**listado.component.html:**

<div fxLayout="row wrap"

    fxLayout.xs="column"

    fxLayoutAlign="center"

    fxLayoutGap="20px">

    <div fxFlex="20"

        fxFlex.lg="15"

        fxFlex.sm="30"

        \*ngFor="let heroe of heroes">           <!-- le indicamos que ocupará 20 px. de la columna -->

       <app-heroe-tarjeta [heroe]="heroe"></app-heroe-tarjeta>

    </div>

</div>

Configuramos en *Styles* de *heroe-tarjeta.component.ts* el formato para las imágenes.

**heroe-tarjeta.component.ts:**

@Component({

  selector: 'app-heroe-tarjeta',

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

  styles: [`

    mat-card {

      margin-top: 20px

    }

  `]

})

export class HeroeTarjetaComponent {

**Cap. 193:**

Utilizamos un Pipe para mostrar las imágenes.

**heroe.tarjeta.component.html:**

 <mat-card>

    <mat-card-header>

        <mat-card-title>{{ heroe.superhero }}</mat-card-title>

        <mat-card-subtitle>{{ heroe.alter\_ego }}</mat-card-subtitle>

    </mat-card-header>

    <!-- ponemos en comentario

        <img mat-card-image src="assets/heroes/{{ heroe.id }}.jpg">

    -->

    <img mat-card-image [src]="heroe | imagen">

Creamos la carpeta PIPES dentro de HEROES.

Adentro generamos el PIPE:

$ng g pipe heroes/pipes/imagen --stip-tests

Se genera este archivo.

**imagen.pipe.ts:**

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

@Pipe({

  name: 'imagen'

})

export class ImagenPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {

    return null;

  }

}

Realizamos los cambios del archivo y queda de esta manera:

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

import { Heroe } from '../interfaces/heroe.interface';

@Pipe({

  name: 'imagen'

})

export class ImagenPipe implements PipeTransform {

  transform( heroe: Heroe ): string {

    return `assets/heroes/${ heroe.id }.jpg`;

    // esto debe retornar:  assets/heroes/dc-batman.jpg

  }

}

**Cap. 194:**

Configuramos los botones de LEER MAS y EDITAR

**heroe-tarjeta.component.html:**

<mat-card-actions>          <!-- espacio reservado para botones -->

        <button mat-button color="warn"

            [routerLink]="[ '/heroes', heroe.id ]">

            Leer mas...

        </button>

        <button mat-button color="info"

            [routerLink]="[ '/heroes/editar', heroe.id ]">

            Editar

        </button>

    </mat-card-actions>

En héroe.component necesitamos inyectar la función para leer el URL

**heroe.component.ts: (heroes/pages/heroe)**

export class HeroeComponent implements OnInit {

  constructor( private activatedRoute: ActivatedRoute ) { }

  ngOnInit(): void {

    this.activatedRoute.params

      .subscribe( ({ id }) => console.log( id ))

  }

}

**Cap. 195:**

Componente de *Angular Material:*

**Grid list**

**Progres spinner**

Siempre en la pestaña API

import {MatGridListModule} from '@angular/material/grid-list';

import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';

Actualizamos:

**material.module.ts:**

import { MatButtonModule } from '@angular/material/button';

import { MatCardModule } from '@angular/material/card';

import { MatGridListModule } from '@angular/material/grid-list';

import { MatIconModule } from '@angular/material/icon';

import { MatListModule } from '@angular/material/list';

import { MatProgressSpinnerModule } from '@angular/material/progress-spinner';

import { MatSidenavModule } from '@angular/material/sidenav';

import { MatToolbarModule } from '@angular/material/toolbar';

@NgModule({

    exports: [

        MatButtonModule,

        MatCardModule,

        MatGridListModule,

        MatIconModule,

        MatListModule,

        MatProgressSpinnerModule,

        MatSidenavModule,

        MatToolbarModule

Agregamos un nuevo servicio:

**heroes.service.ts:**

export class HeroesService {

  constructor( private http: HttpClient ) { }

  getHeroes(): Observable<Heroe[]> {

    return this.http.get<Heroe[]>('http://localhost:3000/heroes');

  }

  getHeroesPorId( id: string ): Observable<Heroe> {

    return this.http.get<Heroe>(`http://localhost:3000/heroes/${ id }`);

  }

}

**heroe.component.ts:**

Inyectar en el constructor el servicio *heroeService*, agregamos el *.pipe* y modificamos el *subscribe*

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

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

import { switchMap } from 'rxjs';

import { Heroe } from '../../interfaces/heroe.interface';

import { HeroesService } from '../../services/heroes.service';

...

...

export class HeroeComponent implements OnInit {

  heroe!: Heroe;

  constructor( private activatedRoute: ActivatedRoute,

              private heoresService: HeroesService) { }

  ngOnInit(): void {

    this.activatedRoute.params

      .pipe(

        switchMap( ({ id }) => this.heoresService.getHeroesPorId(id) )

      )

      .subscribe( heroe => this.heroe = heroe );

  }

}

Entonces el *heroe.component.html* queda de esta manera

**heroe.component.html:**

<mat-grid-list cols="1" \*ngIf="!heroe; else divHeroe">

    <mat-grid-tile>

        <mat-spinner></mat-spinner>

    </mat-grid-tile>

</mat-grid-list>

<ng-template #divHeroe>

    <div fxLayout="row"

        fxLayout.xs="column"

        fxLayoutGap="30px">

        <div fxFlex="50">

            <h1>{{ heroe.superhero }} <small>- {{ heroe.alter\_ego }}</small></h1>

            <mat-divider></mat-divider>

            <br>

            <img [src]="heroe | imagen">

        </div>

        <div fxFlex="50">

            <h1>{{ heroe.publisher }}</h1>

            <mat-list>

                <mat-list-item>{{ heroe.first\_appearance }}</mat-list-item>

                <mat-list-item>{{ heroe.characters }}</mat-list-item>

                <mat-list-item>{{ heroe.publisher }}</mat-list-item>

                <mat-list-item>{{ heroe.alter\_ego }}</mat-list-item>

            </mat-list>

            <br>

            <button mat-button color="warn">

                Regresar

            </button>

        </div>

    </div>

</ng-template>

Programamos el botón *regresar* para que vuelva al listado, para eso importamos el *Router*, creamos el método *regresar.*

**heroe.component.ts:**

constructor( private activatedRoute: ActivatedRoute,

              private heoresService: HeroesService,

              private router: Router) { }

  ngOnInit(): void {

    this.activatedRoute.params

      .pipe(

        switchMap( ({ id }) => this.heoresService.getHeroesPorId(id) )

      )

      .subscribe( heroe => this.heroe = heroe );

  }

  regresar() {

    this.router.navigate(['/heroes/listado']);

  }

}

Implementamos en el html

**heroe.component.html:**

<button mat-button

                color="warn"

                (click)="regresar()">

                Regresar

</button>

**CAMBIANDO EL HOST DE DESARROLLO A PRODUCCIÓN CON VARIABLES DE ENTORNO**

Vamos al archivo *src/enviroments/enviroments.prod.ts* y *enviroments.ts*, allí pegamos el path que se encuentra en nuestro archivo de servicios, el cual será reemplazado (el path sin la / al final).

**heroes.services.ts:**

export class HeroesService {

  constructor( private http: HttpClient ) { }

  getHeroes(): Observable<Heroe[]> {

    return this.http.get<Heroe[]>('http://localhost:3000/heroes');

  }

  getHeroesPorId( id: string ): Observable<Heroe> {

    return this.http.get<Heroe>(`http://localhost:3000/heroes/${ id }`);

  }

}

Ahora creamos la variable URL para desarrollo :

**enviroment.ts:**

export const environment = {

  production: false,

  baseUrl: 'http://localhost:3000'

};

La misma variable para producción, ya con la dirección correcta de nuestra API

**enviroment.prod.ts:**

export const environment = {

  production: true,

  baseUrl: 'http://direccion-correcta/api'

};

Ya podemos reemplazar la variable creada, en nuestro servicio, creamos la variable de entorno, ATENDEMOS que al importar el *enviromet* no nos traiga el de PRODUCCION.

**heroes.service.ts:**

import { environment } from '../../../environments/environment.prod'; \*\*ATENCION\*\*

@Injectable({

  providedIn: 'root'

})

export class HeroesService {

  private baseUrl: string = environment.baseUrl;

  constructor( private http: HttpClient ) { }

  getHeroes(): Observable<Heroe[]> {

    return this.http.get<Heroe[]>(`${ this.baseUrl }/heroes`);

  }

  getHeroesPorId( id: string ): Observable<Heroe> {

    return this.http.get<Heroe>(`${ this.baseUrl }/heroes/${ id }`);

  }

}

Pie de página