

Objetivo crear una Aplicación Web de múltiples capas integrando un Backend programado en C# con acceso a datos mediante EntityFramework, controladores de WebApi2, y un Frontend con vistas en Html, Bootstrap y código Typescript con Angular, todo esto hosteado en Internet Information Server (IIS).

Nuestro proyecto estará compuesto por un menú que nos permitirá navegar entre una página de inicio, una página de consulta sobre la tabla ArticulosFamilias y una página que nos permitirá realizar un ABMC sobre la tabla Articulos.

En el proceso de desarrollo vamos a iniciar por el frontend en el cual trabajaremos inicialmente con datos hardcoded, luego trabajaremos contra un backend ya funcional; y finalmente desarrollaremos nuestro propio backend.

Versión Final del Proyecto, sitio completo:

- <https://pymes2021.azurewebsites.net>
- <https://pav2.azurewebsites.net>

Version Final del Proyecto, codigo fuente frontend:

- <https://stackblitz.com/edit/pymes2021>

Requisitos tener instalado:

- Visual studio Code (frontend)
 - o Instalar en Visual Code la extensión “Angular Language Service” para facilitar la escritura/depuración del código con Angular
 - o Alternativamente trabajaremos con el editor online <https://stackblitz.com/> al cual podemos loguearnos con un usuario registrado en <https://github.com/>
- Visual studio (backend)
- Node.js y npm para trabajar con Angular
 - o CLI Angular 13

Nota revisar las versiones con los comandos de consola:

- `node -v` (16+)
- `npm -v` (8+)
- `ng version` (13.2)

**** Visual Studio, test dependencias revisadas 10/03/2022**

- angular core 13.2
- bootstrap@4.6
 - instala automáticamente jquery@1.9.1
 - instala automáticamente popper.js@1.16.1
- font-awesome (en esta libreria ya el nombre significa version @4.7.0, otras versiones tienen otro nombre)
- @ng-bootstrap/ng-bootstrap@11 (para Angular 13 y Bootstrap 4)

Archivo package.json

```
{  
  "private": true,  
  "dependencies": {  
    "@angular/animations": "~13.2.0",  
    "@angular/common": "~13.2.0",  
    "@angular/compiler": "~13.2.0",  
    "@angular/core": "~13.2.0",  
    "@angular/forms": "~13.2.0",  
    "@angular/platform-browser": "~13.2.0",  
    "@angular/platform-browser-dynamic": "~13.2.0",  
    "@angular/router": "~13.2.0",  
    "@ng-bootstrap/ng-bootstrap": "^11.0.0",  
    "bootstrap": "4.6",  
    "font-awesome": "^4.7.0",  
    "rxjs": "~7.5.0",  
    "tslib": "^2.3.0",  
    "zone.js": "~0.11.4"  
  },  
  "devDependencies": {  
    "@angular-devkit/build-angular": "~13.2.3",  
    "@angular/cli": "~13.2.3",  
    "@angular/compiler-cli": "~13.2.0",  
    "@types/jasmine": "~3.10.0",  
    "@types/node": "^12.11.1",  
    "jasmine-core": "~4.0.0",  
    "karma": "~6.3.0",  
    "karma-chrome-launcher": "~3.1.0",  
    "karma-coverage": "~2.1.0",  
    "karma-jasmine": "~4.0.0",  
    "karma-jasmine-html-reporter": "~1.7.0",  
    "typescript": "~4.5.2"  
  }  
}
```

**** Proyecto online - stackblitz; test dependencias revisadas 10/03/2022**

<https://stackblitz.com/edit/angular-ivy-hv4qrl>

- angular core 13.2
 - bootstrap@4.6
- sugiere instalar:



- font-awesome@4.7.0
 - @ng-bootstrap/ng-bootstrap@11 (para Angular 13 y Bootstrap 4)
- sugiere instalar:



mas angular-cli, typescript....

Archivo package.json:

{ package.json X

```
1 {
2   "name": "angular",
3   "version": "0.0.0",
4   "private": true,
5   "dependencies": {
6     "@angular/animations": "13.2.6",
7     "@angular/common": "13.2.6",
8     "@angular/compiler": "13.2.6",
9     "@angular/compiler-cli": "13.2.6",
10    "@angular/core": "13.2.6",
11    "@angular/forms": "13.2.6",
12    "@angular/localize": "^13.0.0",
13    "@angular/platform-browser": "13.2.6",
14    "@angular/platform-browser-dynamic": "13.2.6",
15    "@angular/router": "13.2.6",
16    "@ng-bootstrap/ng-bootstrap": "11",
17    "bootstrap": "4.6.1",
18    "font-awesome": "^4.7.0",
19    "jquery": "3.6.0",
20    "popper.js": "1.16.1",
21    "rxjs": "7.5.5",
22    "tslib": "2.3.1",
23    "typescript": ">=4.4.2 <4.6",
24    "zone.js": "0.11.5"
25  },
```

Etapas

- Crear proyecto
 - agregar bootstrap y fontawesome (y sus dependencias)
- Crear componente Inicio
- Crear componente ArticulosFamilias
 - Crear interface html: un título que indique el nombre del componente y una tabla con filas que muestre los registro de la tabla ArticulosFamilias. (datos harcodeados en el html)
 - Crear la interface ArticuloFamila, representará un registro de la tabla ArticulosFamilias, cada campo estará mapeado a una propiedad del mismo tipo de datos.
 - Crear array ArticulosFamilias, mientras no tengamos comunicación con el servidor, nos representara los registros de la tabla ArticulosFamilias.
 - Reemplazar tabla estática de la interface Html, con los datos del array ArticulosFamilias
 - Crear servicio mockArticuloFamiliaService
 - recuperar desde el servicio el array ArticulosFamilias (luego se recuperará desde el backend) mediante un método llamado “get()”
 - consumir dicho servicio desde el componente ArticulosFamilias
 - Crear servicio ArticulosFamiliaService
 - Implementar el método Buscar() mediante httpClient contra el servidor de api ya existente: <https://pav2.azurewebsites.net/api>
 - consumir dicho servicio desde el componente ArticulosFamilias, reemplazando a mockArticuloFamiliaService
- Crear componente Menu
 - Integrar menú para poder navegar entre los componentes Inicio y ArticulosFamilias
- Crear componente Articulos
 - Crear interface html
 - Crear formulario “FormBusqueda”: con la cual poder establecer los parametros para buscar Articulos por los campos: Nombre y Activo.
 - Crear tabla-html que muestre los resultados de la Búsqueda, en la cual se verán algunos de los campos de la tabla Articulos
 - Crear formulario “formRegistro” en donde poder agregar/ver/editar todos los campos de un registro en particular de la tabla Articulos
- Crear la interface Articulo, representará un registro de la tabla Articulos, cada campo estará mapeado a una propiedad del mismo tipo de datos.
 - Crear array Articulos, mientras no tengamos comunicación con el servidor, nos representará los registros de la tabla Articulos.
- Crear un servicio mockArticulosService
- Iniciar el enlace a datos del componente ArticulosComponent (reactive forms)
- Implementar la funcionalidad del componente Articulos para el ABM, desarrollando los siguientes métodos:
 - Buscar con los parámetros Nombre y Activo: buscar los registros en el servidor que coincidan con los parámetros.
 - cargar los resultados en la tabla-html

- Agregar: presentar en el formulario "FormRegistro" un registro vacío para permitir cargar un Alta.
- BuscarPorId: buscar los datos de un registro en el servidor según un Id
 - cargar los resultados en el formulario "FormRegistro" y los muestra.
- Editar: hace uso del método BuscarPorId y permite modificar los campos.
- Consultar: hace uso del método BuscarPorId y no permite modificar los campos.
- Grabar: graba los cambios en el servidor luego de un Alta o Edición
- Implementar Validaciones de los datos a ingresar en el ABM
 - Datos requeridos
 - Longitud mínima, máxima
 - Expresión regular: números, importes, fechas
- Implementar componentes de BootstrapUI
 - paginador
 - Formulario Modal -> Alert, Confirm, Bloquear/Desbloquear Pantalla
- Implementar Validación de Formularios
- Implementar funcionalidad avanzada
 - servicio ModalDialog
 - Alert, Confirm
 - Bloquear/Desbloquear Pantalla
 - Interceptores http
 - Captura y display de errores.
 - Control de errores -> alert con msj error

Comenzando el proyecto con Angular (Etapa 1)

1. Inicialmente vamos a crear nuestro proyecto con comandos de CLI Angular, para lo cual nos vamos a la ventana de comandos (ctrl+r cmd) y nos ubicamos en C:\Users\MiUsuario y allí ejecutaremos el comando: ng new pymes2021 (<https://stackoverflow.com/questions/52331625/how-to-create-a-specific-version-of-an-angular-project-using-cli>)
 - a. Se iniciará el proceso de creación del proyecto y responderemos como sigue a las siguientes preguntas:
 - i. ? Would you like to add Angular routing? **No**
 - ii. ? Which stylesheet format would you like to use? **CSS**
 - b. El proceso finalizará luego de algunos minutos y luego de muchos mensajes informativos de todas las acciones realizadas.
 - c. Mediante el explorar de archivo podremos examinar la carpeta creada por el proyecto y todas las subcarpetas y archivos relacionados a la estructura del mismo.
2. Para verificar la funcionalidad la plantilla inicial del proyecto recién creado, nos ubicamos en dicha carpeta y podemos ejecutarlo y abrirlo en el explorador con el siguiente comando de consola: C:\Users\MiUsuario\pymes2021>ng serve --open
Tener en cuenta que debo estar ubicado dentro de la carpeta del proyecto recién creado.
 - a. Podemos detener el servidor de la aplicación node/angular, estando ubicados en la ventana de comando, pulsando Ctrl+C o cerrando la misma.
3. Abra la aplicación Visual Studio Code y siga estos pasos:
 - a. Con la opción Archivo-> Abrir Carpeta seleccione la carpeta raíz de nuestro proyecto:
C:\Users\MiUsuario\pymes2021
Esto permitirá editar todo el código fuente de nuestro proyecto.
 - b. Dentro de los archivos de la aplicación busque el archivo app.component.html que define la vista inicial de nuestro proyecto y reemplace el texto: "is running" por: "esta ejecutándose".
 - c. Luego grabe los cambios y vea como en el explorador se actualiza automáticamente el cambio.
4. Ahora vamos a realizar los puntos 2 y 3 en editor online:
 - a. Ingresamos a la pagina <https://stackblitz.com/>
 - b. Clickeamos la opción: START NEW APP
 - c. Clickeamos la opción: Angular Typescript
 - d. Listo!! Nuestra plantilla de proyecto está generada y vemos su vista previa!
5. Realice el punto 4 en el proyecto de stackblitz
reemplace el texto: "Start editing to see some magic happen :)"
por: (su traducción o algo similar)

Primer componente: Inicio

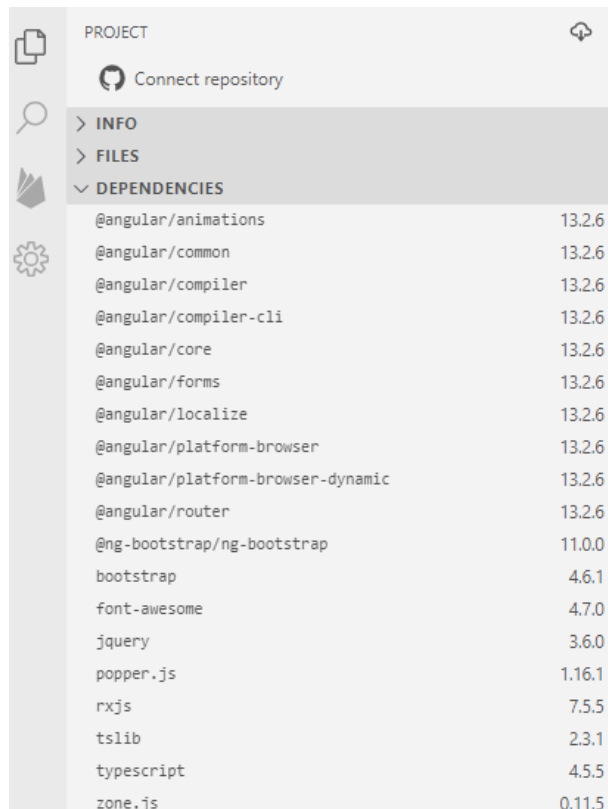
- Eliminar todo el contenido de app_component.html y agregar el siguiente código html

```
<div>
  <h1>Pymes Demo - 2021</h1>
  <p>Este ejemplo está desarrollado con las siguientes tecnologías: </p>
  <p>Backend: VisualStudio 2019, Aplicacion Web con WebApi2, EntityFramework 6
, multiples capas en C#.
  <p>Frontend: Single Page Aplicacion, HTML, CSS, Bootstrap, Typescript y Angu
lar.</p>
  <button type="button">Ver ABMC Articulos </button>
</div>
```

- Agregar al proyecto las librerías de Bootstrap:
 - npm install jquery
 - npm install popper.js
 - npm install bootstrap
 - npm install bootstrap@4.6 (opcionalmente puedo especificar la version)
- Agregar al proyecto las librerías de iconos Font-Awesome
 - npm install font-awesome (esta es la version 4.7 no hace falta especificarla)

Nota: todos los paquetes deben instalarse estando ubicados en la carpeta raíz del proyecto.

**** en StackBlitz las librerías/paquetes se agregan en el panel de la izquierda en la pestaña DEPENDENCIES, al finalizar la misma usar el input que dice “enter package name”**



| PROJECT | |
|-----------------------------------|--------|
| Connect repository | |
| > INFO | |
| > FILES | |
| ▼ DEPENDENCIES | |
| @angular/animations | 13.2.6 |
| @angular/common | 13.2.6 |
| @angular/compiler | 13.2.6 |
| @angular/compiler-cli | 13.2.6 |
| @angular/core | 13.2.6 |
| @angular/forms | 13.2.6 |
| @angular/localize | 13.2.6 |
| @angular/platform-browser | 13.2.6 |
| @angular/platform-browser-dynamic | 13.2.6 |
| @angular/router | 13.2.6 |
| @ng-bootstrap/ng-bootstrap | 11.0.0 |
| bootstrap | 4.6.1 |
| font-awesome | 4.7.0 |
| jquery | 3.6.0 |
| popper.js | 1.16.1 |
| rxjs | 7.5.5 |
| tslib | 2.3.1 |
| typescript | 4.5.5 |
| zone.js | 0.11.5 |

- Primero detenga el servidor; para que estas librerías estén disponibles en nuestro proyecto, debemos indicarle a Angular que queremos incluirlas al mismo, agregando su rutas de acceso en el archivo angular.json, como hijas del array "Build" (cuidado que hay otros styles y scripts, en el mismo archivo pero en otros arrays)

```
"styles": [  
  "node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "node_modules/font-awesome/css/font-awesome.css",  
  "src/styles.css"  
],  
"scripts": [  
  "node_modules/jquery/dist/jquery.min.js",  
  "node_modules/popper.js/dist/umd/popper.min.js",  
  "node_modules/bootstrap/dist/js/bootstrap.min.js"  
]
```

Nota: otra alternativa para referenciar bootstrap seria:

Agregar la siguiente línea al archivo styles.css, para que esté disponible la librería de bootstrap en nuestros estilos (ver alternativas:

<https://www.techiediaries.com/angular-bootstrap/>)

```
@import "~bootstrap/dist/css/bootstrap.css"
```

- Agregamos una línea al archivo tsconfig.json (configuración de TypeScript)

```
"angularCompilerOptions": {  
  "enableI18nLegacyMessageIdFormat": false,  
  "strictInjectionParameters": true,  
  "strictInputAccessModifiers": true,  
  "strictTemplates": true,  
  "strictPropertyInitialization" : false  
}
```

- Ahora vamos a modificar nuestro html para hacer uso de Bootstrap y FontAwesome

```
<div class="jumbotron">  
  <h1>Pymes Demo - 2021</h1>  
  <p>Este ejemplo está desarrollado con las siguientes tecnologías: </p>  
<p>Backend: VisualStudio 2019, Aplicacion Web con WebApi2, EntityFramework 6  
, multiples capas en C#.  
<p>Frontend: Single Page Application, HTML, CSS, Bootstrap, Typescript y Angular.</p>
```

```
<button type="button" class="btn btn-lg btn-primary" >
  <i class="fa fa-search"> </i>
  Ver ABMC Articulos
</button>
</div>
```

- En app.component.ts dentro de la clase “AppComponent” agregar la propiedad Titulo:
Titulo= 'Pymes Demo - 2021';
- En app.component.html reemplazar el literal html por una evaluación de angular:
Reemplazar: <h1>Pymes Demo - 2021</h1>
Por: <h1>{{Titulo}}</h1>

Hasta aquí hemos usado el componente raíz de la aplicación: app.component para nuestra página de inicio, pero en realidad lo que deberíamos hacer es crear al menos un componente específico para cada página de nuestra aplicación....

- Vamos a crear el nuevo componente para nuestra página de inicio y para tener mejor ordenados nuestros componentes lo vamos a hacer dentro de una nueva carpeta denominada “components” (será hija de /pymes2021/src/app)
 - ng generate component components/Inicio
o en forma más abreviada: ng g c components/Inicio
Nota: observe que en el archivo app.module.ts, se importo la clase y se agrego el componente recién creado al array declarations del @NgModule (Stackblitz al 01/05/2022 no hace esta última observación de agregarlo en el módulo)
 - copiamos el código html desde app-component.html hacia inicio.component.html
 - copiamos el la variable titulo desde app-component.ts hacia inicio.component.ts
- Ahora modificamos el app-component.html para que muestre el componente inicio, reemplazamos todo su html simplemente con el selector del componente:

```
<app-inicio></app-inicio>
```

- Desde el explorador comprobamos que se carga la página definida en el html del componente inicio: inicio.component.html
 - Para lo cual ejecutamos la aplicación con: ng serve -o

Componente ArticulosFamilias (Etapa 2)

Ahora vamos a crear el segundo componente de nuestra aplicación que se llamará ArticulosFamilias y servirá para listar los datos de la tabla ArticulosFamilias, simplemente será una tabla html que nos mostrará los dos campos de la tabla ArticulosFamilias

- Vamos a crear el componente ArticulosFamilias

- ng g c components/ArticulosFamilias

Nota: observe que en el archivo app.module.ts, se agregó el import de la clase y el componente recién creado al array declarations del `@NgModule` (solo en visual studio, no en stackblitz)

- copiamos en artículos-familias.component.html el siguiente html en donde definimos una tabla html en donde tenemos harcodeados 2 registros

```
<h3>ArticulosFamilias</h3>
<div>
  <table class="table table-bordered table-striped">
    <tr>
      <th class="text-center">IdArticuloFamilia</th>
      <th class="text-center">Nombre</th>
    </tr>
    <tr>
      <td>1</td>
      <td>Accesorios</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Audio</td>
    </tr>
  </table>
</div>
```

- Ahora modificamos el app-component.html para que muestre el componente articulosfamilias, reemplazamos todo su html simplemente con el selector del componente artículos-familias.

```
<app-articulos-familias></app-articulos-familias>
```

- Desde el explorador comprobamos que se carga la página definida en el html del componente ArticulosFamilias: articulos-familias.component.html

- Para lo cual ejecutamos la aplicación con el comando: `ng serve -o`
- Ahora vamos a definir una interface de typescript que nos represente el objeto `ArticuloFamilia` con sus propiedades. Como hicimos con los componentes agruparemos las interfaces con esta funcionalidad en una nueva carpeta denominada "models".
 - Para lo cual ejecutaremos el comando: `ng generate interface models/ArticuloFamilia`
o abreviado: `ng g interface models/ArticuloFamilia`
 - Agregaremos sus propiedades y quedará de la siguiente manera:

```
export interface ArticuloFamilia {  
  IdArticuloFamilia: number;  
  Nombre: string;  
}
```

- También en el mismo archivo de interface definiremos un array, que contenga un conjunto de `ArticuloFamilia` para poder trabajar con el html, que por ahora tendremos harcodeados o mockeados en typescript, pero que luego traeremos del servidor para mostrarlos en esta página. El archivo `articulo-familia.ts` quedará de la siguiente manera:

```
export interface ArticuloFamilia {  
  IdArticuloFamilia: number;  
  Nombre: string;  
}  
  
export const ArticulosFamilias: ArticuloFamilia[] = [  
  { IdArticuloFamilia: 1, Nombre: "Accesorios" },  
  { IdArticuloFamilia: 2, Nombre: "Audio" },  
  { IdArticuloFamilia: 3, Nombre: "Celulares" },  
  { IdArticuloFamilia: 4, Nombre: "Cuidado Personal" },  
  { IdArticuloFamilia: 5, Nombre: "Dvd" },  
  { IdArticuloFamilia: 6, Nombre: "Fotografia" },  
  { IdArticuloFamilia: 7, Nombre: "Frio-Calor" },  
  { IdArticuloFamilia: 8, Nombre: "Gps" },  
  { IdArticuloFamilia: 9, Nombre: "Informatica" },  
  { IdArticuloFamilia: 10, Nombre: "Led - Lcd" }  
];
```

- A continuación vamos a modificar el componente `articulos-familias` para que desde su typescript (`articulos-familias.component.ts`) se pueda acceder al array de `ArticulosFamilias` recién creados

- Al inicio del archivo, Importamos la clase y el array de `ArticulosFamilias`:

```
import { ArticuloFamilia, ArticulosFamilias } from  
'src/app/models/articulo-familia';
```

Nota: si estamos usando stackblitz las rutas de los imports deben ser relativas para que no de error, entonces quedaría así:

```
import { ArticuloFamilia, ArticulosFamilias } from
'../../models/articulo-familia';
```

- Dentro de la clase ArticuloFamiliaComponent (en articulos-familias.component.ts) agregamos una propiedad que contenga el array de articulosFamilias que luego va ser recorrida (*ngFor) en el html para generar la tabla y una variable Titulo para mostrar arriba de la tabla:

```
Items = ArticulosFamilias;
Titulo = "Articulos Familias"
```

- El codigo del completo de articulos-familias.component.ts quedaria asi:

```
import { Component, OnInit } from '@angular/core';
import { ArticulosFamilias } from
'../src/app/models/articulo-familia'

@Component({
  selector: 'app-articulos-familias',
  templateUrl: './articulos-familias.component.html',
  styleUrls: ['./articulos-familias.component.css']
})
export class ArticulosFamiliasComponent implements OnInit {

  Items = ArticulosFamilias;
  Titulo = "Articulos Familias"
  constructor() { }

  ngOnInit() {
  }

}
```

- Luego modificamos articulos-familias.component.html para que muestre la propiedad Titulo y mediante la directiva *ngFor recorra el array Items y se dibuje la tabla:

```
<h3>{{Titulo}}</h3>
<div>
  <table class="table table-bordered table-striped">
    <thead>
      <tr>
        <th class="text-center">IdArticuloFamilia</th>
        <th class="text-center">Nombre</th>
```

```
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let item of Items">
      <td>{{item.IdArticuloFamilia}}</td>
      <td>{{item.Nombre | uppercase}}</td>
    </tr>
  </tbody>
</table>
</div>
```

- Ahora probamos los cambios realizado ejecutando la aplicación:
 - comando: ng serve -o
- Los componentes solo deberían manejar la relación con la plantilla html y mediante servicios recibir/enviar datos desde/hacia el servidor (o como hasta ahora mockeados). Para ir hacia ese concepto, seguidamente creamos el servicio MockArticulosFamilias, análogamente como hicimos anteriormente lo haremos dentro de una carpeta "services" donde agrupamos los servicios de la aplicación.
 - comando: ng g s services/MockArticulosFamilias

Nota: observe que en el caso de los servicios, estos NO son agregados al array declarations del @NgModule.

- Usamos la librería rx.js para poder simular una llamada asíncrona, tal cual como será cuando se invoque la webapi del backend.

```
import { of } from "rxjs";
```

- Crearemos un método "get" que devuelva el array ArticulosFamlias
- El código del completo de mock-articulos-familias.service.ts quedaria asi:

```
import { Injectable } from '@angular/core';
import { of } from "rxjs";
import { ArticulosFamilias } from "../models/articulo-familia";

@Injectable({
  providedIn: 'root'
})
export class MockArticulosFamiliasService {
  constructor() {}
  get() {
    return of(ArticulosFamilias);
  }
}
```

```
}
```

- Ahora modificamos el componente ArticulosFamilias para que consuma el nuevo servicio y recupere desde allí el array de ArticulosFamilias. Vea el método del constructor, en donde por inyección de dependencia accedemos al servicio.

El código completo de articulos-familias.component.ts quedaría así:

```
import { Component, OnInit } from "@angular/core";
import { ArticuloFamilia } from "../../models/articulo-familia";
import { MockArticulosFamiliasService } from "../../services/mock-articulos-familias.service";

@Component({
  selector: "app-articulos-familias",
  templateUrl: "./articulos-familias.component.html",
  styleUrls: ["./articulos-familias.component.css"]
})
export class ArticulosFamiliasComponent implements OnInit {
  Titulo = "Articulos Familias";
  Items: ArticuloFamilia[] = [];

  constructor(
    private articulosFamiliasService: MockArticulosFamiliasService
  ) {}

  ngOnInit() {
    this.GetFamiliasArticulos();
  }

  GetFamiliasArticulos() {
    this.articulosFamiliasService.get()
      .subscribe((res: ArticuloFamilia[]) => {
        this.Items = res;
      });
  }
}
```

- Ahora vamos a trabajar con datos provenientes de nuestro BackEnd para ello haremos uso de la clase http de angular:
 - En el módulo de nuestra aplicación (app.module.ts) importar la clase HttpClientModule del paquete donde está definido:

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

- También en el módulo agregarla a @NgModule.imports

```
imports: [  
  HttpClientModule,  
  ...
```

- A continuación definiremos un nuevo servicio llamado ArticulosFamilias:
 - comando: `ng g s services/ArticulosFamilias`

Este tendrá por objetivo solicitar al servidor, mediante una llamada a su webapi, los datos de la tabla ArticulosFamilias, sólo implementará el método Get. Reemplace todo el código del mismo con el siguiente:

```
import { Injectable } from "@angular/core";  
import {  
  HttpClient  
} from "@angular/common/http";  
import { of } from "rxjs";  
import { ArticuloFamilia } from "../models/articulo-familia";  
  
@Injectable({  
  providedIn: "root"  
})  
export class ArticulosFamiliasService {  
  resourceUrl: string;  
  constructor(private httpClient: HttpClient) {  
    this.resourceUrl = "https://pav2.azurewebsites.net/api/ArticulosFamilias/";  
  }  
  
  get() {  
    return this.httpClient.get<ArticuloFamilia[]>(this.resourceUrl);  
  }  
}
```

** verifique que la variable resourceUrl que apunta al servidor es la adecuada para su entorno.

- Ahora modificaremos el componente ArticulosFamilias (código typescript) para que consuma este nuevo servicio que reemplaza a MockArticulosFamilias:
 - importar servicio

```
import { ArticulosFamiliasService } from "../services/articulos-familias.service";
```

- modificar constructor

```
constructor(  

```



```
//private articulosFamiliasService: MockArticulosFamiliasService  
private articulosFamiliasService: ArticulosFamiliasService  
}
```

- compruebe que con los cambios realizados los datos se traen desde el servidor.

COMPONENTE MENU (Etapa 3)

Para poder navegar entre las diferentes páginas de nuestra aplicación, hasta ahora representada por los componentes Inicio y ArticulosFamilias vamos a crear un nuevo componente llamado “Menu” que nos permitirá implementar dicha funcionalidad.

- Inicialmente agregamos el paquete con la funcionalidad de ruteo que nos ofrece angular
 - comando: `npm install @angular/router`
** los proyectos de StackBlitz ya tienen esta dependencia incluida por defecto.

- En el módulo de nuestra aplicación (app.module.ts)
 - importamos los objetos que usaremos de la librería recién instalada:

```
import { RouterModule } from '@angular/router';  
import { APP_BASE_HREF } from '@angular/common';
```

- Agregamos la definición de los paths de nuestro menú al array “imports”, la cual expresa la relación de las url con los componentes a mostrar:

```
RouterModule.forRoot([  
  { path: '', redirectTo: '/inicio', pathMatch: 'full' },  
  { path: 'inicio', component: InicioComponent },  
  { path: 'articulosfamilias', component: ArticulosFamiliasComponent }  
])
```

** en path usar minúsculas es case sensitive

- Agregamos BaseRef del router al array “providers” del módulo, esta representa el prefijo de URL que debe conservarse al generar y reconocer las URL:

```
{ provide: APP_BASE_HREF, useValue: '/' }
```

Nota: como alternativa al punto anterior:

- En el head del index.html agregamos la etiqueta
`<base href="/">`
- Creamos componente Menu
 - comando: `ng g c components/Menu`
 - copiamos en menu.component.html el siguiente html, que sacamos de los ejemplos de Bootstrap (<https://getbootstrap.com/docs/4.6/examples/navbar-fixed/>), solo tomamos la

etiqueta nav, y personalizamos sus elementos.

```
<nav class="navbar navbar-dark bg-dark navbar-expand-lg">
  <nav class="navbar navbar-light bg-light text-white">
    PYMES
  </nav>

  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a class="nav-link" href="#">Inicio <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#" title="Gestionar Articulos">Articulos</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#" title="Listado Articulos Familias">ArticulosFamilias</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
          Informes
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="#">Informe de Ventas</a>
          <a class="dropdown-item" href="#">Informe de Compras</a>
          <div class="dropdown-divider"></div>
          <a class="dropdown-item" href="#">Informe de Devoluciones de Clientes</a>
          <a class="dropdown-item" href="#">Informe de Devoluciones de Proveedores</a>
        </div>
      </li>
      <li class="nav-item routerLinkActive="active">
        <a class="nav-link" href="#">Acerca de</a>
      </li>
    </ul>

    <ul class="navbar-nav">
```

```
<li class="nav-item"><a class="nav-link" href="">Bienvenido: Carlo  
s Villan</a></li>  
<li class="nav-item"><a class="nav-link" href="">Cerrar Session</a  
></li>  
</ul>  
</div>  
</nav>
```

- En el html, reemplazamos la propiedad href del html, por la propiedad de angular routerLink
ej: routerLink="/inicio"
** en routerLink usar minúsculas es case sensitive
- Finalmente modificamos el app-component.html para que muestre el componente menú y debajo del mismo pondremos la directiva <router-outlet> que usa el router de angular para mostrar el componente seleccionado en el menú-html con la opción routerLink según la definición en el módulo:

```
<app-menu></app-menu>  
<div class="container" id='divBody'>  
  <router-outlet></router-outlet>  
</div>
```

Note que hemos agregado un div contenedor a la directiva <router-outlet> para modificar su estilo con bootstrap y desde el archivo app-component.css:

```
/* Bajar el contenido del body porque la fixed navbar ocupa el top de la pantalla */  
#divBody {  
  min-height: 75rem;  
  padding-top: 4.5rem;  
}
```

COMPONENTE ARTICULOS (Etapa 4)

- Iniciamos creando el componente con el comando:
 - comando: `ng g c components/Articulos`
 - A continuación vamos a modificar el componente menú para poder navegar al nuevo componente articulos en el archivo **menu.component.html**
 - modificamos el html para incluir el nuevo link

```
<a class="nav-link" routerLink="/articulos" title="Gestionar Articulos">Artículos</a>
```

- modificamos `app.module.ts` para incluir la nueva ruta del menú:

```
{ path: 'articulos', component: ArticulosComponent }
```

- También se puede navegar a un componente específico desde un botón, asignando la propiedad `routerLink`, a modo de ejemplo, vamos a modificar el botón de la componente Inicio **inicio.component.html**, para que navegue hacia la página de artículos.

```
<button type="button" class="btn btn-lg btn-primary" routerLink="/articulos">
```

- Crear interface html en el archivo **articulos.component.html**
Nuestro componente tendrá varios bloques de código html, para las distintas necesidades de nuestro ABMC:

- Un Titulo, junto con la acción actual del ABMC

```
<h3>{{Titulo}} <small>{{TituloAccionABMC[AccionABMC]}}</small></h3>
```

- Un formulario que permitirá establecer una serie de campos mediante los cuales filtrar y buscar en el servidor los registros que cumplan dicho criterio. Los campos que usaremos para filtrar son: Nombre (descripción del artículo) y Activo (booleano que indica si el artículo está activo o no, se usa para la baja lógica).

```
<!-- Formulario Filtro de búsqueda, se ve cuando la AccionABMC es "L"(busqueda/listado)-->
<form name="FormBusqueda" class="bg-light" >
  <div class="form-group row">
    <label class="col-form-label col-sm-2 offset-sm-1">
      Nombre:
    </label>
```

```

<div class="col-sm-3">
  <input type="text" name="Nombre" class="form-control" />
</div>

<label class="col-form-label col-sm-2 offset-sm-1">
  Activo:
</label>

<div class="col-sm-3">
  <select class="form-control" name="Activo" >
    <option *ngFor="let opt of OpcionesActivo">
      {{ opt.Nombre }}
    </option>
  </select>
</div>
</div>

<!-- Botones -->
<div class="row justify-content-center">
  <button type="button" class="btn btn-primary"
(click)="Pagina=1;Buscar();">
    <i class="fa fa-search"> </i> Buscar
  </button>

  &nbsp;

  <button type="button" class="btn btn-primary" (click)="Agregar()">
    <i class="fa fa-plus"> </i>
    Agregar
  </button>
</div>
</form>

```

- Tabla-html con los registros obtenidos del servidor luego de la búsqueda, debajo de la misma se incluye un contador de registros obtenidos, un paginador y un botón para imprimir y finalmente un mensaje para avisar cuando no se encuentren registros según el filtro actual.

```

<!-- Tabla de resultados de búsqueda y Paginador-->
<div id="divTablaResultados">

  <table class="table table-bordered table-striped">
    <thead>
      <tr>

```

```
<th class="text-center">Nombre</th>
<th class="text-center">Precio</th>
<th class="text-center">Stock</th>
<th class="text-center">Fecha de Alta</th>
<th class="text-center">Activo</th>
<th class="text-center text-nowrap">Acciones</th>
</tr>
</thead>
<tbody>
  <tr *ngFor="let item of Items">
    <td>{{item.Nombre | uppercase}}</td>
    <td class="text-right">{{item.Precio | currency}}</td>
    <td class="text-right">{{item.Stock}}</td>
    <td>{{item.FechaAlta | date: 'dd/MM/yyyy'}}</td>
    <td>{{item.Activo ? 'SI' : 'NO'}}</td>
    <td class="text-center text-nowrap">
      <button type="button" class="btn btn-sm btn-outline-primary"
title="Consultar" (click)="Consultar(item)">
        <i class="fa fa-eye"></i>
      </button>
      <button type="button" class="btn btn-sm btn-outline-primary"
title="Modificar" (click)="Modificar(item)">
        <i class="fa fa-pencil"></i>
      </button>
      <button type="button" class="btn btn-sm btn-outline-{{ item.Activo ?
'danger':'success'}}"
        title="{{item.Activo ? 'Desactivar':'Activar'}}"
(click)="ActivarDesactivar(item)">
        <i class="fa fa-{{item.Activo ? 'times':'check'}}"></i>
      </button>
    </td>
  </tr>
</tbody>
</table>
<!-- Paginador-->
<div id="divPaginador">
  <div class="row bg-light">
    <div class="col">
      <span class="label label-default">Registros: {{RegistrosTotal}}</span>
    </div>
    <div class="col text-center">
      Directiva paginador aqui
    </div>
  </div>
```

```

        <div class="col text-right">
            <button type="button" class="btn btn-primary"
(click)="ImprimirListado()">
                <i class="fa fa-print"></i>
                Imprimir
            </button>
        </div>
    </div>
</div>
</div>

<!--No se encontraron registros-->
<div id="divMsjFormBusqueda" class="alert alert-info">
    <i class="fa fa-exclamation-sign"></i>{{Mensajes['SD']}}
    <!-- texto: No se encontraron registros -->
</div>

```

- Un formulario que permitirá agregar/consultar/modificar todos los campos de un registro en particular.

```

<!-- Registro en Alta,Modificacion o Consulta -->
<form name='FormRegistro' >

    <fieldset>
        <!--campo nombre-->
        <div class="form-group row">
            <label class="col-form-label col-sm-2 offset-sm-2" for="Nombre">
                Nombre <span class="text-danger">*</span>
            </label>
            <div class="col-sm-6">
                <input type="text" class="form-control" />
            </div>
        </div>

        <!--campo precio-->
        <div class="row form-group">
            <label class="col-form-label col-sm-2 offset-sm-2" for="Precio">
                Precio <span class="text-danger">*</span>
            </label>
            <div class="col-sm-6">
                <input type="text" name="Precio" class="form-control" />
            </div>
        </div>
    </fieldset>

```



```
</div>

<!--campo stock-->
<div class="form-group row">
  <label class="col-form-label col-sm-2 offset-sm-2" for="Stock">
    Stock <span class="text-danger">*</span>
  </label>
  <div class="col-sm-6">
    <input type="text" name="Stock" class="form-control" />
  </div>
</div>

<!--campo codigodebarra-->
<div class="form-group row">
  <label class="col-form-label col-sm-2 offset-sm-2" for="CodigoDeBarra">
    Codigo De Barra <span class="text-danger">*</span>
  </label>
  <div class="col-sm-6">
    <input type="text" name="CodigoDeBarra" class="form-control" />
  </div>
</div>

<!--campo idarticulofamilia-->
<div class="form-group row">
  <label class="col-form-label col-sm-2 offset-sm-2" for="IdArticuloFamilia">
    Familia <span class="text-danger">*</span>
  </label>
  <div class="col-sm-6">
    <select name="IdArticuloFamilia" class="form-control">
      <option *ngFor="let opt of Familias">
        {{ opt.Nombre }}
      </option>
    </select>
  </div>
</div>

<!--campo fechaalta-->
<div class="form-group row">
  <label class="col-form-label col-sm-2 offset-sm-2">
    Fecha de alta <span class="text-danger">*</span>
  </label>
```

```
<div class="col-sm-6">
  <input class="form-control"
    name="FechaAlta" />
</div>
</div>

<!--campo activo-->
<div class="form-group row">
  <label class="col-form-label col-sm-2 offset-sm-2" for="Activo">Activo</label>
  <div class="col-sm-6">
    <select name="Activo" [disabled]="true" class="form-control">
      <option *ngFor="let opt of OpcionesActivo">
        {{ opt.Nombre }}
      </option>
    </select>
  </div>
</div>
</fieldset>

<!-- Botones Grabar, Cancelar/Volver' -->
<div class="row justify-content-center">
  <button type="button" class="btn btn-primary" (click)="Grabar()">
    <i class="fa fa-check"></i> Grabar
  </button>
  &nbsp;
  <button type="button" class="btn btn-warning" (click)="Volver()">
    <i class="fa fa-undo"></i> Volver/Cancelar
  </button>
</div>

<!--texto: Revisar los datos ingresados...-->
<div id="divMsjFormRegistro" class="row alert alert-danger">
  <i class="fa fa-exclamation-sign"></i>{{Mensajes['RD']}}
</div>

</form>
```

- Ya completo el html, le agregaremos unas líneas de estilo en el archivo `articulos.component.css`:

```
form {
```

```
padding: 10px;
margin-bottom: 1em;
}

form button {
  text-align: center;
  margin: 1em;
}
```

- Ahora vamos a definir una interface de typescript que nos representará de forma tipada cada uno de los objetos artículos que por ahora tendremos harcodeados o mockeados en typescript, pero que luego traeremos del servidor para mostrarlos en esta página. Para lo cual ejecutaremos el comando: `ng generate interface models/Articulo` o abreviado: `ng g interface models/Articulo`
 - Agregaremos sus propiedades y quedará de la siguiente manera:

```
export interface Articulo {
  IdArticulo: number;
  Nombre: string;
  Precio: number;
  CodigoDeBarra: string;
  IdArticuloFamilia: number;
  Stock: number;
  FechaAlta: string;
  Activo: boolean;
};
```

- También en el mismo archivo de clase definiremos un array, que contenga un conjunto de Artículos para poder trabajar con el html hasta que escribamos el código para obtenerlo desde el servidor. El archivo artículos.ts quedará de la siguiente manera:

```
export interface Articulo {
  IdArticulo: number;
  Nombre: string;
  Precio: number;
  CodigoDeBarra: string;
  IdArticuloFamilia: number;
  Stock: number;
  FechaAlta: string;
  Activo: boolean;
};

export const Articulos: Articulo[] = [
```

```
{
  IdArticulo: 108,
  Nombre: "Adaptador usb wifi tl-wn722n",
  Precio: 219.0,
 CodigoDeBarra: "0693536405046",
  IdArticuloFamilia: 9,
  Stock: 898,
  FechaAlta: "2017-01-23T00:00:00",
  Activo: false
},
{
  IdArticulo: 139,
  Nombre: "Aire acondicionado daewoo 3200fc dwt23200fc",
  Precio: 5899.0,
 CodigoDeBarra: "0779816944014",
  IdArticuloFamilia: 7,
  Stock: 668,
  FechaAlta: "2017-01-04T00:00:00",
  Activo: true
}
];
```

** para un mejor testing, del ejemplo online puede obtener una versión con más registros.

<https://stackblitz.com/edit/pymes2021?file=src%2Fapp%2Fmodels%2Farticulo.ts>

- para hacer uso desde el componente Articulos del array Articulos recién definido vamos a crear un servicio llamado MockArticulosServicios que emulará los métodos para comunicar el componente con el futuro servidor WebApi.
 - comando: `ng g s services/MockArticulos`
 - Implementaremos los métodos Get, GetById, Post, Put y Delete que interactúan contra el array "Articulos"

```
import { Injectable } from "@angular/core";
import { of } from "rxjs";
import { Articulo, Articulos } from "../models/articulo";

@Injectable({
  providedIn: "root"
})
export class MockArticulosService {
  constructor() {}
  get(Nombre: string, Activo: boolean|null, pagina: number):any {
    var Items = Articulos.filter(item =>
```

```
    // Nombre == null chequea por null y undefined
    // Nombre === null chequea solo por null
    (Nombre == null ||
item.Nombre.toUpperCase().includes(Nombre.toUpperCase()))
    && (Activo == null || item.Activo == Activo)
  );
  //ordenar
  Items = Items.sort( (a,b) => a.Nombre.toUpperCase() >
b.Nombre.toUpperCase() ? 1 : -1 )
  // paginar con slice
  var RegistrosTotal = Items.length;
  var RegistroDesde = (pagina - 1) * 10;
  Items = Items.slice(RegistroDesde, RegistroDesde + 10);
  return of({ Items: Items, RegistrosTotal: RegistrosTotal });
}
// no usamos get con parametros porque javascript no soporta sobrecarga de
metodos!
getById(Id: number) {
  var item: Artículo = Articulos.filter(x => x.IdArticulo === Id)[0];
  return of(item);
}

post(obj: Artículo) {
  obj.IdArticulo = new Date().getTime();

  obj.IdArticuloFamilia = +obj.IdArticuloFamilia; // convierto a numero,
cuando se envia al servidor se hace automaticamente al enlazar las propiedades
del modelo definido en el backend
  obj.Precio = +obj.Precio;
  obj.Stock = +obj.Stock;

  Articulos.push(obj);
  return of(obj);
}

put(Id: number, obj: Artículo) {
  let indice: number =0;
  var items = Articulos.filter(function(item, index) {
    if (item.IdArticulo === Id) {
      indice = index;
    }
  });
}
```

```
    obj.IdArticuloFamilia = +obj.IdArticuloFamilia; // convierto a número,
    cuando se envia al servidor se hace automáticamente al enlazar las propiedades
    del modelo definido en el backend
    obj.Precio = +obj.Precio;
    obj.Stock = +obj.Stock;

    Articulos[indice] = obj;
    return of(obj);
}

delete(Id: number) {
    var items = Articulos.filter(x => x.IdArticulo === Id);
    items[0].Activo = !items[0].Activo;
    return of(items[0]);
}
}
```

Ahora comenzaremos a desarrollar la funcionalidad del componente typescript ("articulos.component.ts"), iniciaremos con esta plantilla con una funcionalidad básica, en donde definiremos una serie de variables y métodos que nos permitirá interactuar con la interface html ya definida:

```
import { Component, OnInit } from "@angular/core";
import { Articulo } from "../../models/articulo";
import { ArticuloFamilia } from "../../models/articulo-familia";
import { MockArticulosService } from "../../services/mock-articulos.service";
import { MockArticulosFamiliasService } from
"../../services/mock-articulos-familias.service";

@Component({
    selector: "app-articulos",
    templateUrl: "./articulos.component.html",
    styleUrls: ["./articulos.component.css"]
})
export class ArticulosComponent implements OnInit {
    Titulo = "Articulos";
    TituloAccionABMC : { [index: string]: string } = {
        A: "(Agregar)",
        B: "(Eliminar)",
```

```
M: "(Modificar)",
C: "(Consultar)",
L: "(Listado)"
};

AccionABMC : string = "L" // inicia en el listado de articulos (buscar con
parametros)

Mensajes = {
  SD: " No se encontraron registros...",
  RD: " Revisar los datos ingresados..."
};

Items: Articulo[]|null = null;
RegistrosTotal: number = 1;
Familias: ArticuloFamilia[]|null = null;
Pagina = 1; // inicia pagina 1

// opciones del combo activo
OpcionesActivo = [
  { Id: null, Nombre: "" },
  { Id: true, Nombre: "SI" },
  { Id: false, Nombre: "NO" }
];

constructor(
  private articulosService: MockArticulosService,
  private articulosFamiliasService: MockArticulosFamiliasService,
) {}

ngOnInit() {
  this.GetFamiliasArticulos();
}

GetFamiliasArticulos() {
  this.articulosFamiliasService.get().subscribe((res:
ArticuloFamilia[]) => {
    this.Familias = res;
  });
}

Agregar() {
```

```
this.AccionABMC = "A";
}

// Buscar segun los filtros, establecidos en FormRegistro
Buscar() {
    this.articulosService
        .get('', null, this.Pagina)
        .subscribe((res: any) => {
            this.Items = res.Items;
            this.RegistrosTotal = res.RegistrosTotal;
        });
}

// Obtengo un registro especifico según el Id
BuscarPorId(Item:Articulo, AccionABMC:string ) {
    window.scroll(0, 0); // ir al incio del scroll
    this.AccionABMC = AccionABMC;
}

Consultar(Item:Articulo) {
    this.BuscarPorId(Item, "C");
}

// comienza la modificacion, luego la confirma con el metodo Grabar
Modificar(Item:Articulo) {
    if (!Item.Activo) {
        alert("No puede modificarse un registro Inactivo.");
        return;
    }
    this.BuscarPorId(Item, "M");
}

// grabar tanto altas como modificaciones
Grabar() {
    alert("Registro Grabado!");
    this.Volver();
}

ActivarDesactivar(Item:Articulo) {
    var resp = confirm(
        "Esta seguro de " +
        (Item.Activo ? "desactivar" : "activar") +
```



```

        " este registro?");
    if (resp === true)
        alert("registro activado/desactivado!");
    }

    // Volver desde Agregar/Modificar
    Volver() {
        this.AccionABMC = "L";
    }

    ImprimirListado() {
        alert('Sin desarrollar...');
    }
}

```

- Observe en el código typescript agregado los siguiente elemento definidos como propiedades/métodos en la clase:
 - un objeto que tiene una serie de propiedades que representan las acciones que pueden hacerse en el abmc:
 TituloAccionABMC = TituloAccionABMC : { [index: string]: string } = {
 - **A: '(Agregar)', B: '(Eliminar)', M: '(Modificar)', C: '(Consultar)', L: '(Listado)'**
 - una propiedad que indica la acción actual que se está ejecutando del abm:
 AccionABMC : string = 'L'; // inicialmente inicia en el listado de artículos
 - Se usará en combinación con **TituloAccionABMC** en el código html para mostrar el título de la acción actual mediante la expresión:
 TituloAccionABMC[AccionABMC]
 - También se usará para poner visible/invisible (*ngIf) los diferentes bloques de html de la interface, según la acción actual.
 - un objeto con los mensaje para informar eventos al usuario:
 Mensajes = {SD: ' No se encontraron registros...', RD: ' Revisar los datos ingresados...' }
 - los métodos: **Agregar(), Buscar(), Consultar(), Modificar(), Grabar(), ActivarDesactivar(), Volver(), Imprimir Listado()** que son invocados desde la vista html por los eventos click y ejecutan las acciones propias del ABMC
 - Los métodos Consultar y Modificar hacen uso de un método interno **BuscarPorId()** que se encargará de buscar los datos de un registro en particular en el servidor.

- A continuación modificaremos la vista para que sólo estén visibles/habilitados los elementos según correspondan a la acción actual del ABMC, definida por la propiedad **AccionABMC**.
 - Reemplazar : `<form name="FormBusqueda" class="bg-light" >`
 Por: `<form name="FormBusqueda" class="bg-light" *ngIf="AccionABMC == 'L'">`
 - Reemplazar: `<div id="divTablaResultados" class="col-sm-12">`
 Por: `<div id="divTablaResultados" *ngIf="AccionABMC == 'L' && Items!=null && Items.length > 0">`
 - Reemplazar: `<div id="divMsjFormBusqueda" class="alert alert-info">`
 Por: `<div id="divMsjFormBusqueda" class="alert alert-info" *ngIf="AccionABMC == 'L' && Items!=null && Items.length== 0">`
 - Reemplazar: `<form name="FormRegistro" class="bg-light" >`
 Por: `<form name="FormRegistro" class="bg-light" *ngIf="AccionABMC != 'L'">`
 - Reemplazar: `<fieldset>`
 Por: `<fieldset [disabled]="AccionABMC==='C'">`
- Para que en el botón Grabar se muestre solo cuando este en un alta o modificación agregarle la siguiente directiva:
`*ngIf=" AccionABMC == 'A' || AccionABMC == 'M' "`

```
<button type="button" class="btn btn-primary" (click)="Grabar()"
*ngIf=" AccionABMC == 'A' || AccionABMC == 'M' " >
```

- Para que el botón que tiene el texto: 'Volver/Cancelar'; muestre el texto "Cancelar" cuando quiera salir de un alta o modificación sin grabar, ó muestre el texto 'Volver' cuando esté en una consulta; reemplazar el texto 'Volver / Cancelar' por la siguiente expresión:
`{{AccionABMC == 'A' || AccionABMC == 'M' ? 'Cancelar' : 'Volver'}}`

```
<button type="button" class="btn btn-warning" (click)="Volver()"
  <i class="fa fa-undo"></i> {{AccionABMC == 'A' || AccionABMC ==
'M' ? 'Cancelar' : 'Volver'}}
</button>
```

- Modificar el **"divMsjFormRegistro"** que muestra el mensaje **'Revisar Datos'**, que debe activarse cuando no son válidos los datos, por ahora lo ponemos invisible agregándole la siguiente directiva: `*ngIf="false"`

```
<div id="divMsjFormRegistro" class="row alert alert-danger" *ngIf="false">
```

```
<i class="fa fa-exclamation-sign"></i>{{Mensajes['RD']}}
</div>
```

- Pruebe cómo se comporta el proyecto con los cambios realizados.
- Observe la técnica usada en la tabla de artículos en la columna Activo para que aparezca según el valor de la propiedad item.Activo (true/false) el texto SI/NO

```
<td>{{item.Activo ? 'SI' : 'NO'}}</td>
```

- Observe la técnica usada en la tabla de artículos en el columna acciones para que aparezca según el valor de la propiedad item.Activo (true/false) el icono fa-times o fa-check en color rojo o verde.

```
<button type="button" class="btn btn-sm btn-outline-{{ item.Activo ? 'danger':'success'}}"
  title="{{item.Activo ? 'Desactivar':'Activar'}}" (click)="ActivarDesactivar(item)">
  <i class="fa fa-{{item.Activo ? 'times':'check'}}"></i>
</button>
```

El uso de interpolación “{{...}}” para asignar el atributo “class” en la etiqueta “button” del código anterior podría reemplazarse por el uso de “ngClass”, como se muestra el siguiente código alternativo:

```
<button type="button" class="btn btn-sm" [ngClass]="{'btn-outline-danger': item.Activo,
'btn-outline-success': !item.Activo}"
...
</button>
```

- Observe la técnica usada para cargar el combo FamiliasArticulos con los datos traídos desde el servidor; mediante la directiva *ngFor

```
<select name="IdArticuloFamilia" class="form-control" formControlName="IdArticuloFamilia">
  <option *ngFor="let opt of Familias" >
    {{ opt.Nombre }}
  </option>
</select>
```

ENLACE DE DATOS, REACTIVEFORMS (Etapa 5)

A continuación haciendo uso de los objetos de Angular FormGroup y FormControl crearemos en typescript los dos formularios que nos permitirán enlazar el código a los formularios html "FormBusqueda" y "FormRegistro"

- Inicialmente importamos en app.module.ts de angular la librería en donde están dichos objetos:

```
import { ReactiveFormsModule } from "@angular/forms";
```

- Luego también el módulo lo agregaremos al array "imports":

```
ReactiveFormsModule
```

- Luego en el archivo articulo.component.ts importamos:

```
import { FormGroup, FormControl, Validators } from "@angular/forms";
```

- Ahora ya podemos definir y crear los objetos formularios

```
FormBusqueda = new FormGroup({  
  Nombre: new FormControl(null),  
  Activo: new FormControl(null),  
});  
  
FormRegistro = new FormGroup({  
  IdArticulo: new FormControl(0),  
  Nombre: new FormControl(''),  
  Precio: new FormControl(null),  
  Stock: new FormControl(null),  
  CodigoDeBarra: new FormControl(''),  
  IdArticuloFamilia: new FormControl(''),  
  FechaAlta: new FormControl(''),  
  Activo: new FormControl(true),  
});
```

- Ahora enlazaremos los objetos formularios y sus propiedades con nuestro html

- le agregaremos a los formularios la propiedad FormGroup

```
<form name="FormBusqueda" [formGroup]='FormBusqueda' class="bg-light" *ngIf="AccionABMC == 'L'" >
```

```
<form name="FormRegistro" [formGroup]='FormRegistro' class="bg-light" *ngIf="AccionABMC != 'L'" >
```

- le agregamos a cada campo la propiedad FormControlName
Ej:

```
<input type="text" class="form-control" formControlName="Nombre"/>
```

** repetir esto para enlazar todos los controles de ingreso de datos de cada formulario con la propiedad del objeto formulario que corresponda.

- en el caso particular de los <option> de los <select> agregar también la propiedad [ngValue] para que enlace el Id correspondiente:

- en ambos campos Activo (del FormBusqueda y FormRegistro):

```
<option *ngFor="let opt of OpcionesActivo" [ngValue]="opt.Id">
```

- y en el campo IdArticuloFamilia

```
<option *ngFor="let opt of Familias" [ngValue]="opt.IdArticuloFamilia">
```

A continuación implementaremos alguna funcionalidad en el código

typescript/angular, donde por medio de las propiedades de nuestros objetos FormGroup, accederemos a los valores cargados en los controles html.

- En el archivo articulos.component.ts en el método Agregar le vamos a escribir el siguiente código que permitirá resetear todos los campos del formulario a su valor por defecto para dejarlo listo para el alta:

```
Agregar() {  
  this.AccionABMC = "A";  
  this.FormRegistro.reset({ Activo: true, IdArticulo: 0 });  
}
```

- En el archivo articulos.component.ts agregar la siguiente funcionalidad al método Buscar()

```
// Buscar segun los filtros, establecidos en FormRegistro  
Buscar() {  
  this.articulosService  
    .get(this.FormBusqueda.value.Nombre, this.FormBusqueda.value.Activo, this.Pagina)
```

```
.subscribe((res: any) => {  
  this.Items = res.Items;  
  this.RegistrosTotal = res.RegistrosTotal;  
});  
}
```

FUNCIONALIDADES DEL ABMC (Etapa 6)

- **BuscarPorId** (para consultar y/o modificar un registro)
 - **Grabar** (altas y modificaciones)
 - **Activar/Desactivar** (baja lógica)
 - **Implementar Servicio “ArticuloService”**
- En el archivo `articulos.component.ts` agregar la siguiente funcionalidad al método `BuscarPorId()`

```
// Obtengo un registro específico según el Id
BuscarPorId(Item:Articulo, AccionABMC:string ) {

  window.scroll(0, 0); // ir al inicio del scroll

  this.articulosService.getById(Item.IdArticulo).subscribe((res: any) => {

    const itemCopy = { ...res }; // hacemos copia para no modificar el array original del mock

    //formatear fecha de ISO 8601 a string dd/MM/yyyy
    var arrFecha = itemCopy.FechaAlta.substr(0, 10).split("-");
    itemCopy.FechaAlta = arrFecha[2] + "/" + arrFecha[1] + "/" + arrFecha[0];

    this.FormRegistro.patchValue(itemCopy);
    this.AccionABMC = AccionABMC;
  });
}
```

** observe que cuando se recibe el registro del servidor el campo fecha llega desde la webapi convertido en string con el formato ISO 8601 y en el método se lo convierte a string con el formato español dd/MM/yyyy para mostrarlo adecuadamente en el html, más adelante en el método `Grabar()` haremos el proceso inverso.

- En el archivo `articulos.component.ts` agregar la siguiente funcionalidad al método `ActivarDesactivar()`

```
// representa la baja logica
ActivarDesactivar(Item : Articulo) {
  var resp = confirm(
    "Esta seguro de " +
    (Item.Activo ? "desactivar" : "activar") +
    " este registro?");
  if (resp === true)
  {
    this.articulosService
      .delete(Item.IdArticulo)
      .subscribe((res: any) =>
```

```
        this.Buscar()
    );
}
}
```

- En el archivo `articulos.component.ts` agregar la siguiente funcionalidad al método `Grabar()`

```
// grabar tanto altas como modificaciones
Grabar() {

    //hacemos una copia de los datos del formulario, para modificar la fecha y luego enviarlo al servidor
    const itemCopy = { ...this.FormRegistro.value };

    //convertir fecha de string dd/MM/yyyy a ISO para que la entienda webapi
    var arrFecha = itemCopy.FechaAlta.substr(0, 10).split("/");
    if (arrFecha.length == 3)
        itemCopy.FechaAlta =
            new Date(
                arrFecha[2],
                arrFecha[1] - 1,
                arrFecha[0]
            ).toISOString();

    // agregar post
    if (this.AccionABMC == "A") {
        this.articulosService.post(itemCopy).subscribe((res: any) => {
            this.Volver();
            alert('Registro agregado correctamente.');
```

```
            this.Buscar();
        });
    } else {
        // modificar put
        this.articulosService
            .put(itemCopy.IdArticulo, itemCopy)
            .subscribe((res: any) => {
                this.Volver();
                alert('Registro modificado correctamente.');
```

```
                this.Buscar();
            });
    }
}
```


** Observe que antes de enviar el registro al servidor la fecha que estaba en formato string con el formato "dd/MM/yyyy" se convierte a string formato ISO 8601 como vino inicialmente desde el servidor en el método BuscarPorId().

** Observe que según el valor de AccionABMC, se determina si se está grabando un alta o una modificación, para así llamar el metodo post o put respectivamente.

- En el archivo articulos.component.ts vamos a agregar el método GetArticuloFamiliaNombre(Id:number) que mediante su funcionalidad nos permitirá filtrar el array Familias y a partir de un IdArticuloFamilia recuperar el Nombre de la misma:

```
GetArticuloFamiliaNombre(Id:number) {  
  let Nombre = this.Familias.find(x => x.IdArticuloFamilia === Id)?.Nombre;  
  return Nombre;  
}
```

- Y en la tabla html agregaremos la columna Familia, la cual hará uso del método anteriormente definido, observe que se invoca el método pasándole de parámetro el IdArticuloFamilia de la fila actual.

```
....  
<th class="text-center">Familia</th>  
...  
<td>{{GetArticuloFamiliaNombre(item.IdArticuloFamilia)}}</td>  
....
```

- Ahora en forma análoga que hicimos con el servicio ArticulosFamilias que traía los datos del servidor, vamos a crear el servicio Articulos (ArticulosService.ts) Genere el servicio y use el siguiente código para dicho servicio.

```
import { Injectable } from "@angular/core";  
import {  
  HttpClient,  
  HttpHeaders,  
  HttpResponse,  
  HttpParams  
} from "@angular/common/http";  
import { of } from "rxjs";  
import { Articulo } from "../models/articulo";  
  
@Injectable({  
  providedIn: "root"  
})  
export class ArticulosService {
```

```
resourceUrl: string;
constructor(private httpClient: HttpClient) {
  this.resourceUrl = "https://pav2.azurewebsites.net/api/articulos/";
}

get(Nombre: string, Activo: boolean, Pagina: number) {
  let params = new HttpParams();
  if (Nombre != null) {
    params = params.append("Nombre", Nombre);
  }
  if (Activo != null) { // para evitar error de null.ToString()
    params = params.append("Activo", Activo.toString());
  }
  params = params.append("Pagina", Pagina.toString());

  return this.httpClient.get(this.resourceUrl, { params: params });
}

getById(Id: number) {
  return this.httpClient.get(this.resourceUrl + Id);
}

post(obj:Articulo) {
  return this.httpClient.post(this.resourceUrl, obj);
}

put(Id: number, obj:Articulo) {
  return this.httpClient.put(this.resourceUrl + Id, obj);
}

delete(Id: number) {
  return this.httpClient.delete(this.resourceUrl + Id);
}
}
```

** verifique que la variable resourceUrl que apunta al servidor es la adecuada para su entorno.

- Ahora ya creado el servicio, vamos a modificar el componente Articulo en articulos.component.ts: a) importando el servicio y b) modificando el constructor para que consuma el servicio ArticuloService recién creado en lugar MockArticulosService.

```
import { ArticulosService } from "../../services/articulos.service";
```

```
constructor(
```

```
//private articulosService: MockArticulosService,  
//private articulosFamiliasService: MockArticulosFamiliasService,  
private articulosService: ArticulosService,  
private articulosFamiliasService: ArticulosFamiliasService,  
)
```

- Luego compruebe la funcionalidad del Componente, verifique en Chrome, que los datos son recuperados del servidor...

IMPLEMENTANDO DIRECTIVAS DE TERCEROS

Existe un conjunto de componentes de angular desarrollados por terceros (entre muchas otras alternativas) basadas en bootstrap, las que ofrecen una serie de interfaces html /código angular muy útiles para integrar en el desarrollo de nuestras aplicaciones. Podemos ver los componentes ofrecidos en <https://ng-bootstrap.github.io/#/getting-started>

En nuestra aplicación implementaremos de esta librería un componente para paginar los datos traídos desde el servidor. Para lograrlo seguiremos los pasos siguientes:

- Instalar el paquete de angular necesario, desde la version 9, para manejar la localización (para adecuarlo al lenguaje y cultura que corresponda) de los componentes:

```
npm i @angular/localize@13.2
```

Nota: la instalacion deberia actualizar el archivo polyfills.ts, agregandole la siguiente linea (revisar particularmente en stackblitz, sino agregar a mano):

```
import '@angular/localize/init';
```

- instalar la librería/paquete de ng-bootstrap: comando: npm install @ng-bootstrap/ng-bootstrap@11
 - si falla la instalacion anterior, intentar agregando la opción `--legacy-peer-deps`

Dependencias:

| ng-bootstrap | Angular | Bootstrap CSS | Popper |
|-------------------------------------|---------|---------------|--------|
| Show older versions | | | |
| 7.x.x, 8.x.x | 10.0.0 | 4.5.0 | |
| 9.x.x | 11.0.0 | 4.5.0 | |
| 10.x.x | 12.0.0 | 4.5.0 | |
| 11.x.x | 13.0.0 | 4.6.0 | |
| 12.x.x | 13.0.0 | 5.0.0 | 2.10.2 |

- importar los componentes que vayamos a usar de la librería al inicio de app.module.ts

```
import {  
  NgbPaginationModule
```

```
} from "@ng-bootstrap/ng-bootstrap";
```

- en app.module.ts en el array imports de @NgModule, agregar

```
NgbPaginationModule,
```

- en articulo.component.html reemplazar el texto "Directiva paginador aqui" por el selector del componente paginador:

```
<ngb-pagination [(page)]= "Pagina" (pageChange)= "Buscar()" [maxSize]= "10"
[collectionSize]= "RegistrosTotal" [pageSize]= "10"></ngb-pagination>
```

En esta directiva encontramos los siguientes propiedades/eventos:

- "page" es una propiedad de doble enlace y mantiene la selección de la página actual, lo enlazamos a la propiedad del componente "pagina".
- "pageChange" es un evento que se desencadena cuando el usuario cambia de página, que en nuestro caso ejecutará el método Buscar()
- "collectionSize" indica la cantidad total de registros que devuelve la consulta. Lo enlazamos a la propiedad de componente RegistrosTotal que se actualiza cuando ejecutamos el método Buscar()
- "pageSize" define la cantidad de registros que se muestran por página, La relación entre este atributo y el anterior define la cantidad de páginas resultantes.
- "maxSize" define la cantidad de páginas a mostrar en el paginador.

Para entender otras posibles configuraciones del componente consultar la ayuda en la página del proveedor.

VALIDACIÓN DE FORMULARIOS (Etapa 7)

- En nuestro caso vamos a usar validadores sobre formularios reactivos, para lo cual al definir en typescript las propiedades de los formularios, es allí donde agregaremos los validadores.
Inicialmente comenzaremos con validadores de datos requeridos (Validators.required)
) La definición de nuestro formulario "FormRegistro" quedará de la siguiente manera:

```
this.Fthis.FormRegistro = new FormGroup({  
  
  IdArticulo: new FormControl(0),  
  
  Nombre: new FormControl('', [  
    Validators.required  
  ]),  
  
  Precio: new FormControl(null, [  
    Validators.required  
  ]),  
  
  Stock: new FormControl(null, [  
    Validators.required  
  ]),  
  
  CodigoDeBarra: new FormControl('', [  
    Validators.required  
  ]),  
  
  IdArticuloFamilia: new FormControl('', [Validators.required]),  
  
  FechaAlta: new FormControl('', [  
    Validators.required  
  ]),  
  
  Activo: new FormControl(true),
```

```
});
```

- En nuestra interface html, agregar a **todos los controles de ingreso** de datos obligatorios, un mensaje para informar al usuario cuando no se cumple con dicha obligatoriedad. Observe la condición del *ngIf que evalua cuando poner dicho mensaje visible

Ej:

```
<input type="text" class="form-control" formControlName="Nombre" />
<div class="text-danger"
*ngIf="FormRegistro.controls['Nombre'].hasError('required')">
    Dato requerido.
</div>
```

** repetir con todos los datos obligatorios (requeridos)

- Hasta aquí los validadores solo se usan para mostrar un mensaje adecuado al usuario, pero no están impidiendo que los datos con validaciones erróneas sean enviados al servidor mediante el botón “Grabar”. Para lograr este objetivo fundamental de la validación, en código del boton Grabar aplicaremos la siguiente verificación:

```
Grabar() {
    // verificar que los validadores esten OK
    if (this.FormRegistro.invalid) {
        return;
    }
    ...
    // resto del codigo para grabar
    ...
}
```

- Para que todos los datos string ingresados estén en mayúsculas y mejorar el formato de los datos y facilitar las búsquedas (la mayoría de los motores sql son case sensitives), les vamos a agregar una clase de bootstrap para que se vean en mayúsculas (en nuestro ejemplo solo el campo Nombre) Esta transformación es solo visual, luego en el servidor tendríamos que hacer la conversión real a mayúsculas antes de guardar en la base de datos.

```
<input type="text" class="form-control text-uppercase" formControlName="Nombre" />
<div class="text-danger" *ngIf="FormRegistro.controls['Nombre'].hasError('required')">
    Dato requerido.
</div>
```

- A los datos que correspondan a inputs en donde ingresamos texto, le aplicaremos un validador para controlar el mínimo (`Validators.minLength`) y/o máximo (`Validators.maxLength`) caracteres permitidos, teniendo en cuenta la longitud del campo definido en la base de datos. En nuestro caso se lo aplicaremos al campo Nombre en el `FomRegistro`

a. Modificaremos la definición del Control en el Formulario de Typescript:

```
Nombre: new FormControl('', [  
  Validators.required,  
  Validators.minLength(4),  
  Validators.maxLength(55),  
]),
```

- b. En nuestra interface html, debajo del input que corresponda, agregar un mensaje para informar al usuario cuando no se cumple con dicho validador. Observe la condición del `*ngIf` que evalúa cuando poner dicho mensaje visible

```
<div  
  class="text-danger"  
  *ngIf="  
    FormRegistro.controls['Nombre'].hasError('minlength') ||  
    FormRegistro.controls['Nombre'].hasError('maxlength')  
  ">  
  Dato texto, 4 a 55 caracteres.  
</div>
```

- A los controles de los campos Precio, Stock y Código de Barra le agregamos el validador "pattern" que mediante expresiones regulares, controlaremos que solo permita datos numéricos (7, 10 y 13 dígitos respectivamente), y un mensaje de error adecuado en el html (observe la condición del `*ngIf`).

```
Precio: new FormControl(null, [  
  Validators.required,  
  Validators.pattern('[0-9]{1,7}'),
```



```
]),  
  
Stock: new FormControl(null, [  
  
  Validators.required,  
  
  Validators.pattern('[0-9]{1,7}'),  
  
]),  
  
CodigoDeBarra: new FormControl('', [  
  
  Validators.required,  
  
  Validators.pattern('[0-9]{13}'),  
  
]),
```

ej del html del campo Precio:

```
<div class="text-danger" *ngIf="FormRegistro.controls['Precio'].hasError('pattern')">  
  Dato numérico, 1 a 7 dígitos.  
</div>
```

- Al control del campos FechaAlta le agregamos el validador “pattern”, que mediante expresiones regulares, solo permita fechas con el formato “dd/MM/aaaa”; y un mensaje de error adecuado en el html (observe la condición del *ngIf).

```
FechaAlta: new FormControl('', [  
  
  Validators.required,  
  
  Validators.pattern(  
  
    '(0[1-9]|[12][0-9]|3[01])[ -/](0[1-9]|1[012])[ -/](19|20)[0-9]{2}'  
  
  ),  
  
]),
```

```
<div  
  class="text-danger"
```

```
*ngIf="FormRegistro.controls['FechaAlta'].hasError('pattern')"
```

>

Dato fecha, formato dd/mm/aaaa.

```
</div>
```

- Al finalizar el formulario FormRegistro, tenemos un mensaje (div id="divMsjFormRegistro") que debería estar visible solo cuando al menos algún validador de formulario es inválido. Para lograr esto, usaremos la directiva *ngIf (observe la condición). Este último mensaje es importante porque en formularios largos que no están visibles en forma completa en el visor del dispositivo (cuando se hace uso del scroll) sirven para indicar que aun cuando los controles visibles están validos, existen otros inválidos.

```
<!--texto: Revisar los datos ingresados...-->
<div id="divMsjFormRegistro" class="row alert alert-danger" *ngIf="FormRegistro.invalid">
  <i class="fa fa-exclamation-sign"></i>{{Mensajes['RD']}}
</div>
```

- Modificar el formato de la grilla para que el precio tenga formato de número y muestre por defecto 2 decimales.

Reemplace: `<td class="text-right">{{item.Precio}}</td>`

Por: `<td class="text-right">{{item.Precio | currency:'$'}}</td>`

- Mejore la visualización de los mensajes de los errores para que se vean solo cuando los campos hayan sido navegados (touched) o cuando el formulario haya sido intentado enviar (submitted), para esto siga los siguientes pasos:
 - Definir en nuestra clase ArticulosComponent una bandera denominada submitted e inicializarla en false; nos indicará cuando se ha intentado enviar el formulario: en el código del boton "Grabar" ponerla en true.
 - Luego Cambiar las condiciones de visualización **de todos los mensajes de error de los controles**, agregando esta doble condición:

```
<div class="text-danger" *ngIf="(FormRegistro.controls['Nombre'].touched || submitted)
  && FormRegistro.controls['Nombre'].hasError('required')">
  Dato requerido.
</div>
```

** Se muestra si se ha pasado por el control (touched) o se ha intentado enviar el formulario (botón Grabar) y al mismo tiempo tiene error del validador, en este caso "required"

- c. Modifique mensaje (div id="divMsjFormRegistro"), para que se vea solo después de haber intentado enviar el formulario, agregándole la condición de la bandera "submitted"

```
<!--texto: Revisar los datos ingresados...-->
<div id="divMsjFormRegistro" class="row alert alert-danger" *ngIf="submitted && FormRegistro.invalid"
>
  <i class="fa fa-exclamation-sign"></i>{{Mensajes['RD']}}
</div>
```

- d. Modifique la función Agregar() y Modificar() para que reinicie el estado touched de los campos del formulario FormRegistro y de la bandera submitted; agregando las siguiente instrucciones:

```
this.submitted = false;
this.FormRegistro.markAsUntouched(); // funcionalidad ya incluida en el FormRegistro.Reset...
```

- Finalmente para resaltar el borde de los controles con errores usaremos la propiedad de angular ngClass para asignarle cuando corresponda la clase is-invalid de bootstrap, para ello a todos los controles del formulario le agregamos un código similar al del siguiente ejemplo del campo Nombre:

```
<input type="text" class="form-control" formControlName="Nombre"
[ngClass]="{'is-invalid': (FormRegistro.controls['Nombre'].touched || submitted)
&& FormRegistro.controls['Nombre'].errors}" />
```

- La validación de los datos ingresados por el usuario, por razones de seguridad, debe hacerse tanto en el lado del cliente (javascript que corre en el explorador) como en el lado del servidor. Como ejemplo de ello, más adelante veremos, en el BackEnd en la clase del proyecto Datos en el método grabar, como se validan algunos campos obligatorios...

SERVICIOS, INTERCEPTOR (Etapa 8)

Vamos a implementar un servicio que nos permitirá compartir funcionalidad desde los diferentes componentes de nuestra aplicación. Nuestro servicio ofrecerá 3 funcionalidades:

- a) Alert(): similar al alert() de javascript pero con bootstrap y en modo asíncrono
- b) Confirm(): similar al confirm() de javascript pero con bootstrap y en modo asíncrono
- c) Bloquear/DesbloquearPantalla(): formulario modal que evitará que un usuario del sistema llame 2 veces a la misma acción, pensando que no funcionó. Para lo cual se bloqueará la interface html hasta que se complete dicha acción. Un caso típico podría ser clicar 2 veces el botón grabar porque el servidor está lento y el usuario no tiene una retroalimentación de que la acción ya está en curso y tiene que esperar a que termine su ejecución.

1. Crearemos la interface visual que estará asociada a nuestro servicio, para lo cual crearemos el componente ModalDialog.

- a. Le asignaremos el siguiente html:

```
<div class="modal-header" [ngClass]="classHeader">
  <h4 class="modal-title">{{titulo}}</h4>
  <button type="button" class="close" aria-label="Close" (click)="activeModal.dismiss('Cross click')">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
<div class="modal-body">
  <p>
    <i *ngIf="!bloquearPantalla" [ngClass]="falcon"></i>
    {{mensaje}}
  </p>
  <div *ngIf="bloquearPantalla" class="progress" >
    <div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar"
      aria-valuenow="100" aria-valuemin="0" aria-valuemax="100" style="width: 100%"></div>
  </div>
</div>
<div class="modal-footer">
  <button *ngIf="textoBotonTrue!="" type="button" class="btn btn-sm btn-outline-primary"
    data-dismiss="modal" (click)="activeModal.close(true)">{{textoBotonTrue}}</button>
  <button *ngIf="textoBotonFalse!="" type="button" class="btn btn-sm btn-outline-danger"
    data-dismiss="modal" (click)="activeModal.close(false)">{{textoBotonFalse}}</button>
</div>
```

- b. Le asignaremos el siguiente typescript:

```
import { Component, OnInit } from '@angular/core';
import { NgbActiveModal, NgbModal } from '@ng-bootstrap/ng-bootstrap';
```

```
@Component({
  selector: 'app-modal-dialog',
  templateUrl: './modal-dialog.component.html',
  styleUrls: ['./modal-dialog.component.css']
})
export class ModalDialogComponent implements OnInit {

  titulo = "";
  texto = "";
  textoBotonTrue = "";
  textoBotonFalse = "";
  bloquearPantalla = false;
  classHeader = 'bg-success';
  falcon = 'far fa-check-circle';
  mensaje = "";

  ngOnInit(): void {
  }
  constructor(public activeModal: NgbActiveModal) {
    this.bloquearPantalla = false;
  }
  cerrar() {
    this.activeModal.close();
  }
  setTipo(tipo: string = 's') {
    tipo = tipo.toLowerCase();
    switch (tipo) {
      case 's':
        this.classHeader = 'bg-success';
        this.falcon = 'far fa-check-circle';
        break;
      case 'd':
        this.classHeader = 'bg-danger';
        this.falcon = 'fa fa-exclamation-triangle';
        break;
      case 'i':
        this.classHeader = 'bg-info';
        this.falcon = 'fa fa-info-circle';
        break;
      case 'w':
        this.classHeader = 'bg-warning';
        this.falcon = 'fa fa-exclamation-triangle';
        break;
      default:
```

```
this.classHeader = 'bg-success';  
break;  
}  
}  
}
```

** Note que el servicio hace uso de las clases NgbModal y NgbModalRef de la librería ngbootstrap.
por lo cual necesitaremos:

- importar el módulo que contiene dichas clases al inicio de app.module.ts

```
import {  
  NgbPaginationModule,  
  NgbModalModule  
} from "@ng-bootstrap/ng-bootstrap";
```

- agregar en app.module.ts en el array @NgModule.imports

```
imports: [  
  NgbModalModule,  
  ....
```

- Continuando con el componente ModalDialog, como todos los otros componentes (Inicio, ArticulosFamilias, Menu y Articulos), necesitamos importarlo al módulo y agregarlo en el array @NgModule.Declarations. (lo hace en forma automática el “ng new component”)

- Por último como el componente ModalDialog será activado imperativamente (desde código) y no desde su selector html, necesita ser registrado en @NgModule.entryComponents

```
entryComponents: [ModalDialogComponent],
```

2. Crearemos un servicio llamado ModalDialog y le asignaremos el siguiente typescript:

```
import { Injectable } from '@angular/core';  
import { NgbModal, NgbModalRef } from '@ng-bootstrap/ng-bootstrap';  
import { ModalDialogComponent } from '../components/modal-dialog/modal-dialog.component';  
  
@Injectable({  
  providedIn: 'root'  
})  
  
export class ModalDialogService {  
  private contadorBloqueo = 0;
```

```
private modalBloqueoRef: NgbModalRef;

constructor(private ngbModal: NgbModal) { }

/// tipo: Info, Danger, Warning, Success
Alert(mensaje?: string, titulo: string = 'Atencion', tipo: string = 'i') {
  const modalRef = this.ngbModal.open(ModalDialogComponent);
  modalRef.componentInstance.titulo = titulo;
  modalRef.componentInstance.mensaje = mensaje;
  modalRef.componentInstance.textoBotonTrue = "";
  modalRef.componentInstance.textoBotonFalse = "";
  modalRef.componentInstance.setTipo(tipo);
  return modalRef;
}

Confirm(mensaje: string, titulo: string = 'Confirmacion',
  textoBotonTrue: string = 'Aceptar',
  textoBotonFalse: string = 'Cancelar',
  funcionTrue: any,
  funcionFalse: any,
  tipo: string = 'w') {
  const modalRef = this.ngbModal.open(ModalDialogComponent);
  modalRef.componentInstance.mensaje = mensaje;
  modalRef.componentInstance.titulo = titulo;
  modalRef.componentInstance.textoBotonTrue = textoBotonTrue;
  modalRef.componentInstance.textoBotonFalse = textoBotonFalse;
  modalRef.componentInstance.setTipo(tipo);
  modalRef.result.then(x => x ? funcionTrue() : funcionFalse() );
  return modalRef;
}

BloquearPantalla() {
  this.contadorBloqueo++;
  if (this.contadorBloqueo === 1) {
    this.modalBloqueoRef = this.ngbModal.open(ModalDialogComponent, { backdrop: 'static', keyboard: false });
    this.modalBloqueoRef.componentInstance.titulo = 'Atencion';
    this.modalBloqueoRef.componentInstance.mensaje = 'Procesando, espere por favor!';
    this.modalBloqueoRef.componentInstance.textoBotonTrue = "";
    this.modalBloqueoRef.componentInstance.textoBotonFalse = "";
    this.modalBloqueoRef.componentInstance.bloquearPantalla = true;
    this.modalBloqueoRef.componentInstance.setTipo('i');
  }
}

DesbloquearPantalla() {
```

```
this.contadorBloqueo--;  
if (this.contadorBloqueo === 0) {  
  this.modalBloqueoRef.close();  
}  
}  
}
```

** observe e identifique los elementos principales del código.

3. Ahora para usar dicho servicio en el componente Artículo:
 - a. Lo importamos al inicio del mismo:

```
import { ModalDialogService } from "../../services/modal-dialog.service";
```

- b. Recibimos el objeto por inyección en el constructor:

```
constructor(  
  private articulosService: MockArticulosService,  
  private articulosFamiliasService: MockArticulosFamiliasService,  
  private modalDialogService: ModalDialogService  
) {}
```

Ahora haremos uso del servicio recién definido:

- Usar la función Alert() del servicio para reemplazar todas las llamadas a la función javascript alert(), como vemos en el siguiente ejemplo

Reemplazar: alert("No puede modificarse un registro Inactivo.");

Por: this.modalDialogService.Alert("No puede modificarse un registro Inactivo.");

- Usar la función Confirm() del servicio para reemplazar la función typescript confirm().
Note que al ser asíncrona cambiará sintaxis:

```
// representa la baja logica  
ActivarDesactivar(Item:Articulo) {  
  this.modalDialogService.Confirm(  
    "Esta seguro de " +  
    (Item.Activo ? "desactivar" : "activar") +  
    " este registro?",  
    undefined,  
    undefined,  
    undefined,  
    () =>  
      this.articulosService  
        .delete(Item.IdArticulo)  
        .subscribe((res: any) =>  
          this.Buscar()  
        )  
    )  
}
```



```
),  
  null  
);  
}
```

- Usar la función BloquearPantalla(), DesbloquearPantalla() del servicio para evitar que se llame más de una vez a la misma acción por error (se da generalmente cuando estamos esperando la respuesta del servidor, pero nos parece que no funcionó el click) . En este caso la usaremos para el botón Buscar del FormBusqueda que trae los datos del servidor.

Haga las siguientes modificaciones del método Buscar()

```
// Buscar segun los filtros, establecidos en FormRegistro  
Buscar() {  
  this.modalDialogService.BloquearPantalla();  
  this.articulosService  
    .get(this.FormBusqueda.value.Nombre, this.FormBusqueda.value.Activo, this.Pagina)  
    .subscribe((res: any) => {  
      this.Items = res.Items;  
      this.RegistrosTotal = res.RegistrosTotal;  
      this.modalDialogService.DesbloquearPantalla();  
    });  
}
```

IMPLEMENTADO INTERCEPTORES

Ahora vamos un paso más allá implementado el Bloquear/DesbloquearPantalla para que se ejecute siempre que ejecutemos una comunicación con el servidor (llamada ajax). Lo hacemos mediante un interceptor de peticiones ajax.

- Vamos a crear una clase llamada MyInterceptor y como tiene una funcionalidad compartida por toda la aplicación la pondremos en una carpeta llamada “shared” (análogo al criterio de las carpetas components, models, etc).
Le asignaremos el siguiente código:

```
import { Injectable } from '@angular/core';
import {
  HttpEvent,
  HttpRequest,
  HttpHandler,
  HttpInterceptor,
  HttpResponse
} from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, finalize } from 'rxjs/operators';
import { ModalDialogService } from '../services/modal-dialog.service';

@Injectable()
export class MyInterceptor implements HttpInterceptor {
  constructor(private ms: ModalDialogService) {}

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {

    this.ms.BloquearPantalla();

    return next.handle(req).pipe(
      catchError((error: HttpResponse) => {
        // 401 handled in auth.interceptor
        if (error.status !== 401 && error.error && error.error.ExceptionMessage) {
          this.ms.Alert( error.error.ExceptionMessage, 'Error', 'd');
        }
        return throwError(error);
      }),
      finalize( () => this.ms.DesbloquearPantalla()),
    );
  }
}
```

```
}  
}
```

- Lo importamos en el módulo de la aplicación:

```
import { HttpClientModule, HTTP_INTERCEPTORS } from "@angular/common/http";  
import { MyInterceptor } from "../shared/my-interceptor";
```

- y lo agregamos al array Providers del @NgModule, de la manera siguiente:

```
providers: [  
  ...  
  { provide: HTTP_INTERCEPTORS, useClass: MyInterceptor, multi: true }  
]
```

- Observe en el código que antes de cada petición (request) se bloquea la pantalla y luego al finalizar el proceso con o sin error, se desbloquea la pantalla.
 - Compruebe esta funcionalidad sacando el código del método Buscar del Componente Artículo que estaba llamando explícitamente a BloquearPantalla/DesbloquearPantalla y verifique que sigue funcionando. También verifique que ahora funciona con todas las peticiones realizadas contra el servidor: Buscar, BuscarPorId, Grabar, etc.
- (solo el backend en desarrollo: corriendo en visual studio) Observe también que se está capturando posibles errores de las peticiones/respuestas al/del servidor, en la misma, se busca el mensaje de error y si existe, se lo muestra haciendo uso del servicio MatDialog anteriormente definido.
 - Compruebe esta funcionalidad grabando dos Artículos con el mismo nombre, lo cual genera un error controlado en el BackEnd con un mensaje adecuado para el usuario.

BACKEND:

- Para cuando estemos en desarrollo y evitar el error de CORS al querer consumir la webapi desde otro dominio (el frontend difiere del backend) debemos instalar el paquete NuGet Install-Package Microsoft.AspNet.WebApi.Cors y en el archivo WebApiConfig agregar al inicio del método Register las siguientes instrucciones:

```
var cors = new System.Web.Http.Cors.EnableCorsAttribute("*", "*", "*");
config.EnableCors(cors);
```

Ajustes antes de publicar la aplicacion

- En el Backend: por seguridad, cambiar cors para solo habilite el sitio del frontend, y si es el mismo que el backend comentar las lineas que permitan cors.
- En el Backend configurar en web.config la cadena de conexion correcta al base de produccion.
- En el fronted configurar la variable environment.ConexionWebApiProxy con la opcion correcta, indicando la url de la webpi del backend

Publicar en IIS

- Si en Sql server usamos seguridad integrada, dar permiso para base del usuario del Pool de iis, normalmente: IIS
APPPOOL\DefaultAppPool
- habilitar los enlaces http/https
 - usar certificado autofirmado para desarrollo.
- tenemos que hacer una configuración especial del mismo para que el ruteo de angular siga funcionando correctamente.
(<https://medium.com/@srinu.vasu289/routing-issues-with-angular-on-iis-97478a6c47d1>) Dicha configuracion la lograremos modificando el web.config agregando el siguiente codigo:

```
<rewrite>
  <rules>
    <rule name="WebApi Routes" stopProcessing="true">
      <match url="api/*" />
      <action type="None" />
    </rule>
    <rule name="Angular Routes" stopProcessing="true">
      <match url="*" />
      <conditions logicalGrouping="MatchAll">
        <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
        <add input="{REQUEST_FILENAME}" matchType="IsDirectory"
negate="true" />
      </conditions>
    </rule>
  </rules>
</rewrite>
```

```
<action type="Rewrite" url="/" />  
</rule>  
</rules>  
</rewrite>
```

Muchas Gracias!

Correcciones, sugerencias y comentarios: jiglesias.20000@gmail.com