Práctica #6

Objetivo

- Crear un servicio que simule el acceso a un web API de alumnos
- Crear un servicio que encapsule el acceso a las listas de descripciones
- Agregar el comportamiento asíncrono al servicio de alumnos

1) Crear el servicio de Alumnos

Utilizaremos angular-cli para generar el esqueleto del servicio, para ello:

- En la terminal, ejecute el siguiente comando

  ng generate service alumnos --flat --module app

Esto generará las siguientes modificaciones en el proyecto:

```
create src/app/alumnos.service.spec.ts (380 bytes)
create src/app/alumnos.service.ts (113 bytes)
update src/app/app.module.ts (1279 bytes)
```

Puede eliminar el archivo con extensión spec.ts ya que no generaremos test alguno sobre este servicio (el tema de testing está fuera del alcance de este curso)

- Edite el archivo **app.module.ts**, revise que el servicio se haya agregado en el array de **providers** del módulo principal de nuestra aplicación. Angular inyecta cada servicio definido en este array a nivel del módulo.
- Edite el archivo **alumnos.service.ts** y revise la cómo se ha definido la clase.
- Agregue el siguiente import al principio del archivo:

```
import { Alumno } from './alumno';
```

Ahora es necesario ahora mover parte del código que tenemos en el componente app al servicio:

- Edite el archivo **app.comonent.ts**. Mueva la propiedad del array de alumnos al archivo **alumnos.service.ts** dentro de la clase como una propiedad más.
- Convierta la propiedad **alumnos** en privada y agréguele un underscore delante del nombre. El código en el archivo **alumnos.service.ts** deberá quedar así:

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

import { Alumno } from './alumno';

@Injectable({
    providedIn: 'root'
})
export class AlumnosService {

  private alumnos: Alumno[] = [
    {id: 1, nombre: 'Juan', apellido: 'Perez', sexo: 1, perfil: 0,
activo: true},
    {id: 2, nombre: 'Pedro', apellido: 'Garcia', sexo: 1, perfil: 1,
activo: true},
    {id: 3, nombre: 'Ana', apellido: 'Romero', sexo: 0, perfil: 2,
activo: true},
    {id: 4, nombre: 'Maria', apellido: 'Gutierrez', sexo: 0, perfil: 2,
activo: true},
    {id: 5, nombre: 'Esteban', apellido: 'Smith', sexo: 1, perfil: 0,
activo: true}
  ];

  constructor() { }

}
```

Agregue los siguientes métodos a la clase debajo del constructor:

```
GetAll(): Alumno[] {
  return this.alumnos;
}

Get(id: number): Alumno {
    const index = this.alumnos.findIndex( (a) => a.id === id);
   if (index < 0) {
     return null;
   }
   return this.alumnos[index];
}

Add(alumno: Alumno): number {
```

```
  alumno.id = new Date().valueOf(); // "unique" id

  this.alumnos.push(alumno);
  return alumno.id;
}

Delete(id: number): void {

  const index = this.alumnos.findIndex( (a) => a.id === id);
  this.alumnos.splice(index, 1);
}

Update(alumno: Alumno) {

  const index = this.alumnos.findIndex( (a) => a.id === alumno.id);
  if (index >= 0) {
    this.alumnos[index] = alumno;
  } else {
    throw  new Error('alumno inexistente');
  }
}

FindbyNombreOApellido(nombre: string) {
  return this.alumnos
             .filter( a => (a.nombre + ' ' + a.apellido)
             .toLowerCase()
             .indexOf(nombre.toLocaleLowerCase()) >= 0);
}
```

- Revise el cómo hemos implementado los métodos.
- Guarde los cambios del archivo.

2) Utilizar el servicio en el componente app

- Modifique el código del componente **app.component.ts** según lo que esta resaltado a continuación:

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

import { AlumnosService } from './alumnos.service';

import { Alumno } from './alumno';
// import { ItemList } from './ItemList';

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

  alumnos: Alumno[];
  alumnoSeleccionado: Alumno = null;

  constructor(private alumnosService: AlumnosService) {}

  ngOnInit() {
    this.alumnos = this.alumnosService.GetAll();
  }

  Seleccionar(alumno: Alumno): void {
     this.alumnoSeleccionado = alumno;
  }

  FinDeEdicion(): void {
     this.alumnoSeleccionado = null;
  }

}
```

Lo que hemos hecho aquí es lo siguiente:

- Implementamos la interface **OnInit** para poder atrapar el evento de inicialización del componente.
- Hemos inyectado el servicio **Alumnos** en el constructor de la clase.
- En el evento **ngOnInit** hemos llamado a la función **GetAll** que nos retorna el array de alumnos.

- Guarde los cambios y pruebe la aplicación.

3) Implementación del servicio Item-list

Si revisa el código de los componentes **alumnos-lista y alumno-edicion** encontrará que allí están los arrays y funciones para obtener la descripción de las propiedades sexo y perfil. El objetivo de este punto es encapsular todo este código en un servicio.

Para ello haga lo siguiente:

- Genere un servicio llamado **item-list**
- Mueva el código de las definiciones de los arrays de **sexos** y **perfil** que se encuentran en el código de **alumno-edicion.component.ts** a este servicio como propiedades privadas.
- Mueva también las funciones **Sexos y Perfiles** al nuevo servicio.
- Cree dos funciones llamadas **GetSexos()** y **GetPerfiles()** que retornen los arrays privados

El código del servicio **item-list.service.ts** deberá quedar así:

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

import { ItemList } from './itemList';

@Injectable({
    providedIn: 'root'
})
export class ItemListService {

  private sexos: ItemList[] = [ new ItemList( 0, 'Femenino'),
    new ItemList( 1, 'Masculino'),
    new ItemList( 2, 'Otro')
  ];

  private perfiles:  ItemList[] = [ new ItemList( 0, 'Desarrollador'),
    new ItemList( 1, 'IT'),
    new ItemList( 2, 'Power User'),
  ];

  constructor() { }

  Sexos(index: number): string {
    return this.sexos.find(item => item.index === index).descripcion;
  }

  Perfiles(index: number): string {
    return this.perfiles.find(item => item.index === index).descripcion;
  }
```

```
  GetSexos(): ItemList[] {
    return this.sexos;
  }

  GetPerfiles(): ItemList[] {
    return this.perfiles;
  }

}
```

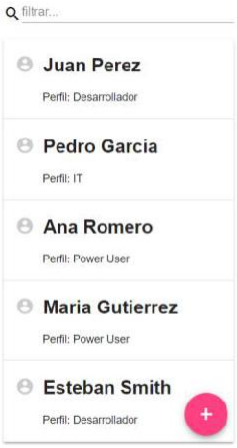
4) Utilización del servicio item-list en los componentes alumnos-lista y alumno-edicion.

Vamos a utilizar este nuevo servicio en los componentes alumnos-lista y alumno-edicion.

- Edite el componente **alumnos-lista.component.ts**
- Elimine todas las funciones y arrays referidos a Perfiles.
- Importe el servicio de **ItemListService**
- Inyecte el servicio de **ItemListService** en el constructor como una propiedad **public ItemListSrv**
- Guarde los cambios
- Edite el archivo **alumnos-lista.component.html**
- Invoque directamente la función **ItemListSrv.Perfiles()** para obtener la descripción del perfil.
- Guarde los cambios y pruebe la aplicación
- Realice lo mismo para perfiles y sexos en el componente **alumno-edicion**
- Guarde los cambios y pruebe la aplicación

5) Agregar la funcionalidad de filtrar alumnos y agregar alumnos

Los siguientes pasos nos llevaran a modificar el componente de **alumnos-lista** para agregar la funcionalidad antedicha. Para tener una idea de hacia donde vamos el componente se mostrará de esta forma luego de las modificaciones que hagamos:



- Edite el archivo **alumnos-lista.component.html** y agregue el siguiente código al principio del mismo, antes del **<mat-card class="lista">**

```
<div class="search-box">
  <mat-icon >search</mat-icon>
  <mat-form-field>
      <input matInput #filtrar placeholder="filtrar..."
      (keyup)="Filtrar(filtrar.value)">
  </mat-form-field>
</div>
```

Este código nos mostrará el ícono de buscar (lupa) y un input text box que disparará la función **Filtrar**() cada vez que se presione una tecla

- Al final del código html agregue la siguiente línea:

```
<button mat-fab class="add-button" (click)="Agregar()">+</button>
```

Aquí agregaremos un fab button (https://material.angular.io/components/button/examples) que se muestra como un circulo con el símbolo + y que dispara en el click del botón la función **Agregar()**

- Guarde los cambios
- Vamos a agregar la funciones **Filtrar() y Agregar().** Dado nuestro patrón de diseño la misma emitirán eventos que serán atrapados por el componente padre. Edite el archivo **alumnos-lista.component.ts**

- Agregue estos eventos antes de constructor:

```
  @Output() AgregarAlumno = new EventEmitter<void>();
  @Output() FiltrarAlumnos = new EventEmitter<string>();
```

Estos eventos deberán ser luego atrapados por el componente padre.

- Al final de la clase agregue las siguientes funciones:

```
  Agregar() {
    this.AgregarAlumno.emit();
  }

  Filtrar(filtro: string) {
    this.FiltrarAlumnos.emit(filtro);
  }
```

Vea como estas funciones trabajan disparando los eventos del componente.

- Guarde los cambios
- Edite el archivo alumnos-lista.componet.css y agregue las siguientes clases al final de este:

```
.search-box {
  display: flex;
  align-items: center;
  justify-content: center;
}

.search-box mat-form-field {
  flex: 1 1 auto;
}

.add-button {
  position: absolute;
  bottom: 8px;
  right: 8px;
  font-size: 32px;
}

:host {
```

```
  position: relative;
}
```

Estas reglas nos permitirán acomodar los elementos visuales tal cual lo queremos (que es el display: flex; y flex: 1 1 auto?)

- Guarde los cambios
- Pruebe la aplicación. El resultado final debe ser similar a la imagen mostrada al principio de este punto

Ahora modificaremos el componente **app** para atrapar y responder a los eventos. Comenzaremos por la funcionalidad de filtrar alumnos en la lista.

- Edite el archivo **app.component.html** y agregue el código para atrapar el evento de filtro del componente **alumnos-lista**:

```
<app-alumnos-lista
    [alumnos] = "alumnos"
    (Seleccion)="Seleccionar($event)"
    [alumnoSeleccionado]="alumnoSeleccionado"
    (FiltrarAlumnos)="FiltrarAlumnos($event)">
</app-alumnos-lista>
```

- Guarde los cambios
- Edite el archivo **app.component.ts** y agregue al final la función **FiltrarAlumnos()**

```
FiltrarAlumnos(filtro: string): void {
  this.alumnos = this.alumnosService.FindbyNombreOApellido(filtro);
}
```

La función es muy simple. Cada vez que se modifique el filtro se disparará esta función que invocará a la función de nuestro servicio que nos retornará un array filtrado y lo asignamos a nuestro array **alumnos** que a su vez está asociado a la propiedad de **Input alumnos** del componente alumnos-lista.

- Guarde los cambios y pruebe la funcionalidad de filtro.

Ahora nos queda implementar la funcionalidad de agregar un alumno. Para ello atraparemos el evento que emite el componente alumnos-lista y crearemos una nueva instancia del objeto

Alumno vacía y la asignaremos a **alumnoSeleccionado** para que el componente **alumno-edicion** la muestre para su edición. Para ello edite el archivo **app.component.html** y atrape el evento:

```
<app-alumnos-lista
    [alumnos] = "alumnos"
    (Seleccion)="Seleccionar($event)"
    [alumnoSeleccionado]="alumnoSeleccionado"
    (FiltrarAlumnos)="FiltrarAlumnos($event)"
    (AgregarAlumno)="AgregarAlumno()">
</app-alumnos-lista>
```

- Guarde los cambios
- Edite el archivo **app.component.ts** y agregue la función **AgregarAlumno()** al final de la clase:

```
AgregarAlumno(): void {
  this.alumnoSeleccionado = new Alumno(0, '', '', 0, true, 0);
}
```

- Guarde los cambios y pruebe la aplicación

Vea la sencillez de la solución utilizando a **alumnoSeleccionado**. Recordemos que esta propiedad se pasa tanto al componente **alumnos-lista** como al componente **alumno-edicion**

Nos falta ahora modificar el componente **alumno-edicion** para que diferencie cuando se está editando un alumno y cuando se está agregando.

- Edite el archivo alumno-edicion.component.html y modifique las siguientes líneas

```
<mat-card *ngIf="alumnoSeleccionado">
  <h2 *ngIf="alumnoSeleccionado.id != 0">Editar Alumno</h2>
  <h2 *ngIf="alumnoSeleccionado.id == 0">Agregar Alumno</h2>

  <form novalidate #f="ngForm">
```

Dependiendo del id del alumnoSeleccionado mostraremos un título u otro

- Agregue a nivel de la carpeta **App** un archivo llamado **operaciones.ts**. En ese archivo agregue el siguiente código:

```
export type Operaciones = 'agregar' | 'editar' | 'eliminar' | 'cancelar';
```

Acabamos de crear un nuevo tipo de dato llamado String Literal Type en TypeScript. Básicamente funcionan como un enum en donde los únicos valores permitidos al tipo Operaciones son los definidos por los string (más información: https://www.typescriptlang.org/docs/handbook/advanced-types.html)

- Guarde los cambios.

Vamos a utilizar el tipo **Operaciones** para informar en el evento de **FinDeEdicion()** si hemos cancelado, agregado o editado un alumno.

- Edite el código del componente alumno-edicion. Agregue el siguiente import:

```
import { Alumno } from '../alumno';
import { ItemList } from '../ItemList';
import { Operaciones } from '../operaciones';

@Component({
```

- Modifique el evento **FinDeEdicion** para informar el resultado de la edición

```
export class AlumnoEdicionComponent implements OnInit {

  @Input() alumnoSeleccionado: Alumno;
  @Output() FinDeEdicion = new EventEmitter<Operaciones>();

  constructor(
```

- Edite las funciones **Regresar() y Guardar()**. La función Regresar() ya no pondrá en nulo la propiedad **alumnoSeleccionado.**

```
  Regresar() {
    // this.alumnoSeleccionado = null;
    this.FinDeEdicion.emit('cancelar');
  }

  Guardar(form: any) {

    Object.keys(form).forEach((key, index) =>
```

```
    this.alumnoSeleccionado[key] = form[key]
  );

  if (this.alumnoSeleccionado.id === 0) {
    this.FinDeEdicion.emit('agregar');
  } else {
    this.FinDeEdicion.emit('editar');
  }
}
```

- Guarde los cambios.

Ahora tendremos que modificar el componente **app** para saber el resultado de la operación y desde este agregar el alumno nuevo utilizando la función **Add** del servicio **AlumnosService**

- Edite el archivo **app.component.html** y edite el evento **FinDeEdicion** en el componente **app-alumno-edicion**

```
<app-alumno-edicion
    [alumnoSeleccionado]="alumnoSeleccionado"
    (FinDeEdicion)="ActualizarAlumno($event)">
</app-alumno-edicion>
```

- Guarde los cambios.
- Edite el archivo **app.component.ts** y agregue el siguiente import:

```
import { Alumno } from './alumno';
import { Operaciones } from './operaciones';

@Component({
```

- Agregue la función **ActualizarAlumno()** al final de la clase:

```
ActualizarAlumno(operacion: Operaciones): void {
  if (operacion === 'agregar') {
    this.alumnosService.Add(this.alumnoSeleccionado);
  }
  this.alumnoSeleccionado = null;
}
```

- Guarde los cambios y pruebe la aplicación.