Desarrollando un CRUD en ANGULAR

Parte 4
Servicios

Mct. Esteban Calabria

Enunciado

En el laboratorio pasado hicimos un los formularios reactivos para nuestro CRUD

Vamos a crear un servicio para nuestro proyecto

¿Qué son los Servicios en Angular?

Los componentes no deben gestionar el acceso a los datos directamente ni operar con datos simulados conscientemente. Su función principal es la de presentar datos, delegando la obtención y el almacenamiento de datos en servicios especializados.

Los servicios que encapsulan lógica y datos que se pueden reutilizar a través de la aplicación y facilitar la compartición de datos y funcionalidades entre componentes.

Beneficios de Usar Servicios

- Reutilización de Código: Evita duplicación de lógica.
- Mantenibilidad: Facilita la gestión y actualización de la lógica de negocio.
- Testabilidad: Se pueden testear de forma aislada.
- Inyección de dependencias

Antes de continuar debe saber lo siguiente

👉👉👉👉👉👉

Capa de Servicios

Inyección de dependencias

Libreria RXJS

Crear un servicio CRUD en memoria

Paso 1

- Crear el servicio
- Programar el Servicio
- Integrarlo con los Componentes

Crear el servicio

Podemos crear el servicio con el comando ng

```
>ng generate service /services/contacto
```

```
CREATE src/app/services/contactos.service.spec.ts (388 bytes)
CREATE src/app/services/contactos.service.ts (147 bytes)
```

Observar el código generado

@Injectable es un decorador que marca una clase como participante en el sistema de inyección de dependencias de Angular. Permite que Angular maneje la creación y provisión de instancias de la clase decorada, facilitando el uso de servicios en diferentes partes de la aplicación.

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

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

  constructor() { }
}
```

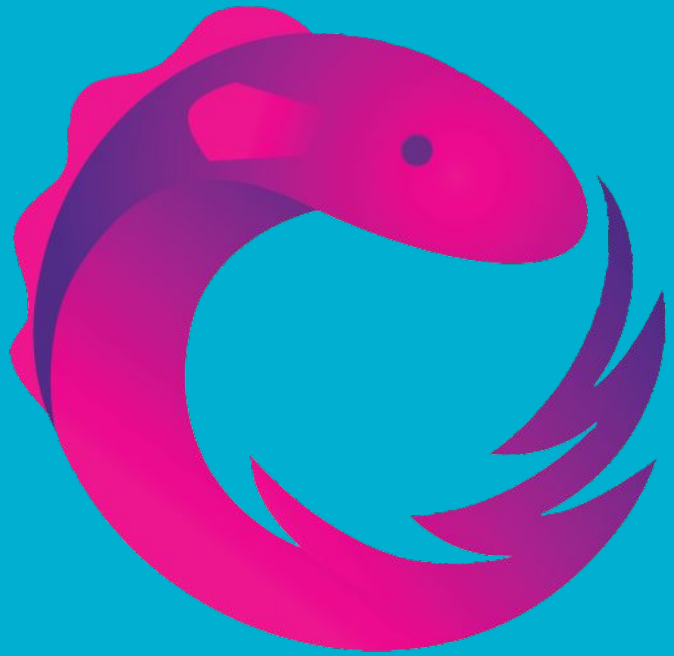
Definamos los métodos necesarios

Observemos Que devuelven observables

```
import { Injectable } from '@angular/core';
import { ContactoModel } from '../models/contacto.model';
import { Observable, of } from 'rxjs';

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

  private contactos: ContactoModel[] = [];

  constructor() {}

  obtenerContactos(): Observable<ContactoModel[]> { ... }

  obtenerContactoPorId(id: number): Observable<ContactoModel | undefined> {...  }

  agregarContacto(contacto: ContactoModel): void {  }

  actualizarContacto(contacto: ContactoModel): void {  }

  eliminarContacto(id: number): void {  }
}
```

Observables

RxJS (Reactive Extensions for JavaScript) es una biblioteca para programación reactiva usando Observables que facilita la composición de secuencias asíncronas y basadas en eventos. RxJS es una parte central del desarrollo en Angular y es fundamental para gestionar operaciones asincrónicas como solicitudes HTTP, eventos de usuario y otros flujos de datos.

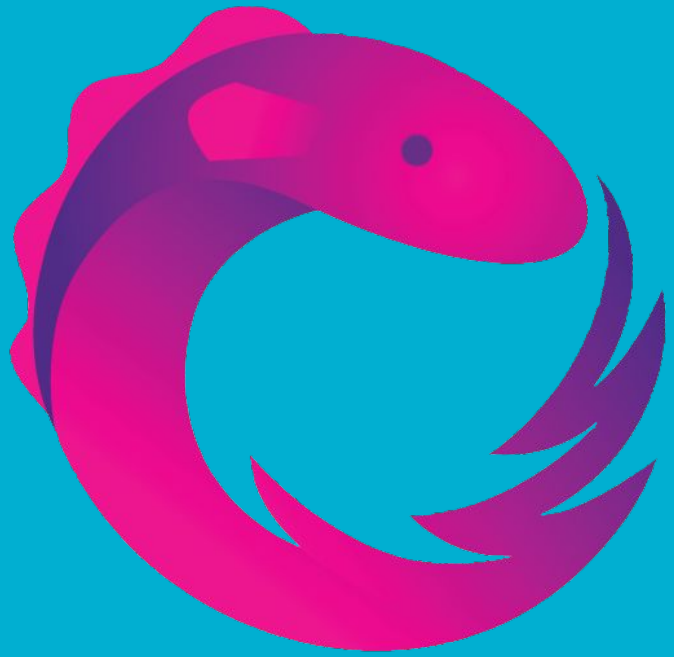


Angular usa por defecto Observables en vez de promesas para llamadas asíncronas

Vamos a suponer que los métodos de los servicios son siempre asincrónicos por si en el futuro resultan serlo

La url de la libreria rxjs es https://rxjs.dev/

- Gestión Avanzada de Asincronía: Facilita el manejo de múltiples eventos asíncronos.
- Composición: Combina flujos de datos de diversas fuentes.
- Transformación: Aplica transformaciones complejas a los datos.
- Reactivo: Ideal para aplicaciones que reaccionan a eventos y cambios de estado.



obtenerContactos

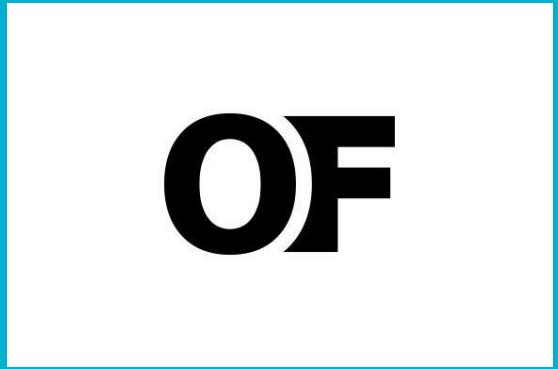
```
obtenerContactos(): Observable<ContactoModel[]> {
    return of(this.contactos);
  }
```

Uso de of en RxJS

El operador of en RxJS es una función de creación de Observables que emite una secuencia de valores y luego completa. Es particularmente útil cuando se desea crear un Observable a partir de un conjunto finito de valores o de valores literales.

Concepto

- Función: of es una función de creación en RxJS que convierte un número variable de argumentos en una secuencia de valores que se emiten de manera síncrona.
- Salida: Produce un Observable que emite los valores proporcionados como argumentos uno tras otro y luego emite una notificación de completado.



obtenerContactoPorId

```
obtenerContactoPorId(id: number): Observable<ContactoModel | undefined> {
    const contacto = this.contactos.find(c => c.id === id);
    return of(contacto);
  }
```

agregarContacto

```
  agregarContacto(contacto: ContactoModel): void {
    this.contactos.push(contacto);
  }
```

actualizarContacto

```
actualizarContacto(contacto: ContactoModel): void {
    const index = this.contactos.findIndex(c => c.id === contacto.id);
    if (index !== -1) {
      this.contactos[index] = contacto;
    }
  }
```

EliminarContacto

```
  eliminarContacto(id: number): void {
    this.contactos = this.contactos.filter(c => c.id !== id);
  }
```

Inyectando el observable a un componente

```
export class CrearContactoComponent {

  form: FormGroup;

  constructor(private fb: FormBuilder,
    private router: Router,
    private service : ContactoService) {
```

Utilizando el contacto

```
export class ListadoContactosComponent {

  contactos: ContactoModel[] = [];

  constructor(private service: ContactoService) {}

  ngOnInit(): void {
    this.service.obtenerContactos().subscribe(contactos => {
      this.contactos = contactos;
    });
  }
}
```

Sigamos Trabajando...