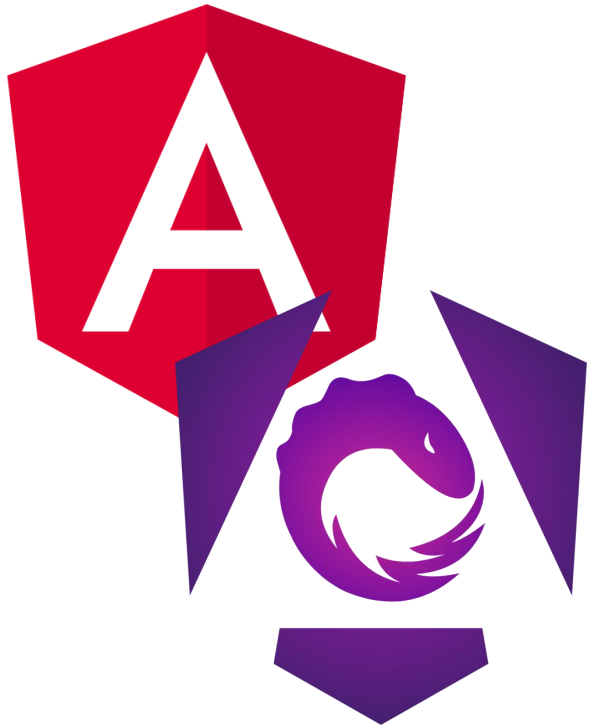


FLUX/REDUX Y NGRX

Angular

ANGULAR

Flux/Redux/NgRx

ANGULAR

- **RxJS**:
  - RxJS es una librería JavaScript para crear programas asíncronos y basados en eventos, mediante el uso de secuencias observables.

- **Flux:**
  - Flux es un patrón diseñado por Facebook para mantener el estado de las aplicaciones de forma coherente. Destinado principalmente a **React**.

- **Redux**:
  - Es una implementación simplificada de **Flux**.

- **NgRx**:
  - Es un sistema de gestión de estado para **Angular** basado en **RxJS** e inspirado en **Redux** (Angular + RxJS + Redux).

ANGULAR

RxJS

ANGULAR

- Evolución de las soluciones asíncronas en JavaScript:
  - Funciones callback
  - Promises
  - Async/Await
  - RxJS

ANGULAR

- RxJS se basa en el uso los patrones ***Observer*** e ***Iterator***, así como en la **programación funcional**.
- Patrón *Observer*:
  - Permite que un objeto (llamado *observable* o *subject*) notifique automáticamente a otros objetos (*observadores*) sus cambios de estado, evitando el acoplamiento del código.
- Patrón *Iterator*:
  - Permite proporcionar una forma uniforme de recorrer (iterar) los elementos de una colección sin exponer su estructura interna.
- Programación funcional:
  - Es un paradigma de programación que se basa en el uso de funciones puras y el tratamiento de los datos como valores inmutables.

ANGULAR

- RxJS. Componentes:
  - **Observable**: Representa un flujo de datos o de eventos que se pueden emitir en algún momento futuro.
  - **Observer**: Son objetos que están "observando" el flujo de datos.
  - **Subscription**: Relación entre *observer* y *observable*. Se puede cancelar.
  - **Operador**: Funciones (programación funcional) que operan sobre los flujos de datos y eventos.
  - **Subject**: Permite comunicar los **observables** a varios *observers* simultáneamente.
  - **Schedulers**: Planificadores para controlar el orden las suscripciones y la velocidad en la emisión de eventos.

ANGULAR

- RxJS. **Creación de observables**.
  - **of**() → Crea un observable a partir de valores individuales → of(1, 2, 3)
  - **from**() → Convierte arrays, promesas o iterables en observables → from([1, 2, 3])
  - **interval**() → Emite números en intervalos de tiempo → interval(1000) → 0, 1, 2, ...
  - **timer**() → Espera un tiempo y luego emite (o repite) valores → timer(2000, 1000)
  - **fromEvent**() → Crea un observable desde un evento del DOM → fromEvent(button, 'click')

ANGULAR

- RxJS. **Creación de observables**. Ejemplos.
  - En el Service de Angular:
    - public miObservable = from("Cadena de caracteres");
    - public miObservable = of("Item 1", "Item 2", "Item 3");
    - public miObservable = interval(1000);

ANGULAR

- RxJS. **Consumo de observables**.
  - Mediante el método **subscribe** (tratamiento simple):

```
miObservable.subscribe(valor=> {
      console.log(valor);
});
```

  - Mediante el método **subscribe** (tratamiento de errores y fin):

```
miObservable.subscribe({
  next: v => console.log("Valor:", v),
  error: e => console.error("Error:", e),
  complete: () => console.log("Listo")
});
```

ANGULAR

- RxJS. **Consumo de observables**.
  - Mediante el pipe **async** de Angular:

```
<span>Wait for it... {{ greeting | async }}</span>
```

  - Convirtiendo el observable a promesa (*promise*) y obteniendo el primero o el último valor:

```
const valor = await
firstValueFrom(miObservable);
const valor = await
lastValueFrom(miObservable);
```

ANGULAR

- RxJS. **Consumo de observables**.
  - Introduciendo una capa de control mediante la función tap → ejecuta una función asociada al evento, pero no forma parte del proceso.

```
miObservable
  .pipe(tap(x => console.log("Recibido:", x)))
  .subscribe();
```

ANGULAR

- RxJS. **Funciones de transformación**.
  - map() → transforma cada valor que emite el Observable.
  - filter() → deja pasar solo los valores que cumplen una condición.
  - switchMap() → cambia a un nuevo Observable y cancela el anterior.
  - merge() → combina varios Observables en uno que emite todo lo que ellos emiten.

ANGULAR

- RxJS. **Cancelación de suscripción**.
  - Asignándole la suscripción a una variable y utilizando el método *unsubscribe*.

```
let suscripcion = this.service_1.miObservable
      .pipe(tap(x=>console.log("Recibido:"+x)))
      .subscribe(valor => {
        console.log(valor);
        this.dato += valor;
});
suscripcion.unsubscribe();
```

ANGULAR

- RxJS. **Consumo de observables**.
  - Mediante el uso de *signals* (a partir de Angular 16) con la función *toSignal*.

```
public mySignal = toSignal(this.service_1.miObservable);
```

ANGULAR

Signals

ANGULAR

- **Signals.**
  - Angular 16+
  - Una *signal* es un *wrapper* que envuelve un valor que notifica a los consumidores interesados cuando ese valor cambia. Las *signals* pueden contener cualquier valor, desde primitivos hasta estructuras de datos complejas.
  - El valor de una *signal* se lee llamando a su función getter, lo que permite a Angular realizar un seguimiento de dónde se utiliza la señal.
  - Las *signals* pueden ser de escritura o de solo lectura.

ANGULAR

- **Signals.**
  - signal() → Crea un estado reactivo mutable (similar a un "state")
  - input() → Reemplazo del decorador @Input()
  - output() → Reemplazo del decorador @Ouput
  - model() → Reemplazo de la directiva [(ngModel)] (*two-ways binding*)
  - computed() → Crea un valor derivado que se recalcula automáticamente
  - effect()→ Ejecuta efectos secundarios cuando cambian signals (reacción automática)
  - viewChild() y viewChildren() → Versión reactiva para acceder a elementos del DOM o componentes hijos en la vista
  - contentChild() y contentChildren() → Versión reactiva para acceder al contenido proyectado (ng-content)

ANGULAR

- **Signals avanzados.**
  - untracked() → Permite leer signals dentro de computed/effect sin crear dependencias reactivas
  - isSignal() → Verifica si un valor es un signal
  - setSignal() → Actualiza un signal desde funciones externas sin llamar .set() directamente (API moderna)
  - toSignal() y toObservable() → Interoperabilidad entre RxJS y signals (Observable ↔ Signal)
  - linkedSignal() → Crea un signal que se sincroniza automáticamente con fuentes externas o valores derivados complejos
  - resource() → Manejo reactivo de cargas asincrónicas (estado + loading + error + datos)

ANGULAR

- **Signals.**
  - **Declaración:**
    - **Signal modificable:**
      - `mi_signal_1 : WritableSignal<number> = signal(0)`
    - **Signal de solo lectura:**
      - `mi_signal_2 : Signal<number> = computed(()=> this.count() * 2)`
  - **Creación**:
    - **Crea un *signal* con un valor inicial (siempre):**
      - `mi_signal_1 : WritableSignal<number> = signal(0)`
    - **Crea un *signal* calculado:**
      - `mi_signal_2 : Signal<number> = computed(()=> this.count() * 2)`

ANGULAR

- **Signals. Ejemplos:**
  - **Model:**
    - TS:
      - public title = **model**<string>(‘’);
      - **this.title()**.toLowerCase());
    - HTML:
      - <input type="text" class=".nes-input" **[(ngModel)]="title"**>

  - **Effect:**

```
effect(() => {
      console.log("El nombre es: " + this.servicio.doubleCount());
});
```

ANGULAR

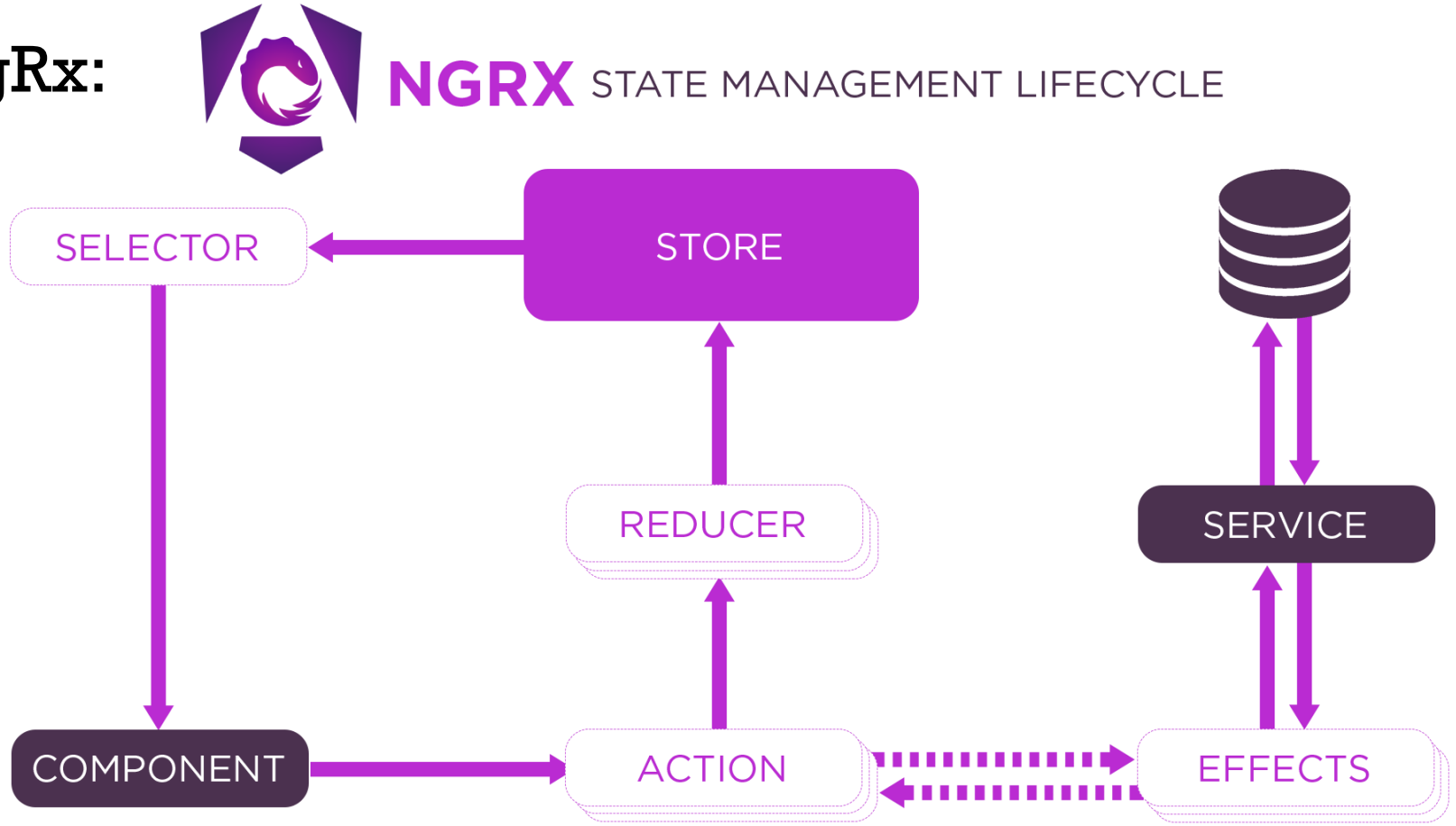
NgRx

ANGULAR

- NgRx:
  - Basado en el patrón REDUX.
  - Los componentes principales de NgRx son:
    - Store (almacena el estado).
    - Acciones (eventos que describen cambios).
    - Reducers (funciones puras que actualizan el estado en función de las acciones).
    - Selectors (funciones que extraen datos específicos del estado).
    - Effects (que manejan operaciones asíncronas como llamadas HTTP).

ANGULAR

- NgRx:



Fuente de la imagen: https://ngrx.io/guide/store

ANGULAR

- NgRx:
  - Alternativas:
    - **NgRx Store tradicional.**
      - **Angular 2+**
      - Estado centralizado en un ***store* único** (árbol de estado global).
      - Acceso reactivo mediante ***observables*** y ***selectors***.
      - Inmutabilidad obligatoria: los ***reducers*** siempre retornan un nuevo estado.
      - Flujo típico: dispatch → reducer → selector → componente.
      - Ideal para aplicaciones grandes y con lógica compleja o efectos asincrónicos.
    - **NgRx SignalStore.**
      - **Angular 16+**
      - Puede haber un ***store* raíz o varios feature stores** (por sección o funcionalidad).
      - Cada *store* creado con ***createSignalStore*** mantiene su propio estado.
      - Está diseñado para funcionar con las primitivas más recientes de Angular, como **signals**, **inject**()y **computed**().
      - Elimina la necesidad de ***actions***, ***reducers*** y ***selectors***.
      - Más **simple**, ***menos boilerplate***, integración directa con *standalone* components.
      - Acceso al estado directo y reactivo, sin necesidad de subscribe ni *async pipe*.
      - Actualización mediante *update*() y selección mediante *select*().

ANGULAR

- NgRx:
  - Instalación:
    - **ng add @ngrx/schematics**@latest
      - Es una herramienta de desarrollo, no de ejecución (no afecta a la ejecución de la aplicación).
      - Solo facilita la creación de archivos de NgRx (actions, reducers, effects, stores).
      - Se puede instalar tanto si se usa @ngrx/store como si se usa @ngrx/signals.
    - **ng add @ngrx/store**@latest **(ng add @ngrx/store**@latest **--no-minimal)**
      - Es el **NgRx clásico** (Redux style).
      - Usa acciones, reducers, selectors, effects.
      - Es ideal para apps grandes, complejas o empresariales donde el estado se comparte entre muchos módulos.
    - **ng add @ngrx/signals**@latest
      - Es la forma de manejar estado en **Angular 16+** (incluido Angular 20).
      - Usa la API de Signals, mucho más **simple** y reactiva sin RxJS ni actions.
      - Menos código, más rendimiento, integración nativa con Angular.
      - **Ideal para apps nuevas** o para estado local/moderadamente global.
      - Se puede combinar con @ngrx/store si se necesita, pero no es obligatorio.
    - **ng add @ngrx/effects**@latest
      - El módulo de NgRx que se usa para manejar efectos secundarios (side effects)

https://ngrx.io/guide/store/install

ANGULAR

- NgRx:
  - Instalación:
    - **ng add @ngrx/store-devtools@latest**
      - Opcional: muestra el estado en el navegador (no es compatible con NgRx SignalStore)
      - Necesita disponer de una extensión en el navegador.
        - https://chromewebstore.google.com/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd

https://ngrx.io/guide/store/install

ANGULAR

- **NgRx Store**. Compontentes:
  - State: Estado almacenado.
  - Actions: Las acciones o eventos que puede realizar o generar un usuario.
  - Reducers: Los elementos que actualizan el estado en función de la acción realizada o evento emitido.
  - Selectors: Permite obtener el estado, ya sea parcial o totalmente.

ANGULAR

- **NgRx Store**.
  - Generación de elementos:
    - Action:
      - ng generate action counter/counter
        - Crea src/app/counter/counter.actions.ts
    - Reducer:
      - ng generate reducer counter/counter
        - Crea src/app/counter/counter.reducer.ts

ANGULAR

- **NgRx Store**. Action Groups.
  - Alternativa a las actions individuales.
  - Agrupa las acciones relacionadas en un solo objeto.
  - Evita duplicación de prefijos.
  - Recomendable.

ANGULAR

- **NgRx Store**. Ejemplo.
  - Actions:

```
import { createActionGroup, emptyProps } from '@ngrx/store';

export const CounterActions = createActionGroup({
  source: 'Counter',
  events: {
    increment: emptyProps(),
    decrement: emptyProps(),
    reset: emptyProps(),
  },
});
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - Reducer:

```
import { createReducer, on } from '@ngrx/store';
import { CounterActions } from './counter.actions';

export const initialState = 0;

export const counterReducer = createReducer(
  initialState,
  on(CounterActions.increment, state => state + 1),
  on(CounterActions.decrement, state => state - 1),
  on(CounterActions.reset, state => 0)
);
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - app.config.ts:

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners,
provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';


import { routes } from './app.routes';
import { provideStore } from '@ngrx/store';
import { counterReducer } from './counter/counter.reducer';


export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    provideStore({ count: counterReducer })
]
};
```

Registro de los reducers

ANGULAR

- **NgRx Store**. Ejemplo.
  - app.ts:

Importación del componente standalone en la aplicación

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { Counter } from "./compontents/counter/counter";


@Component({
  selector: 'app-root',
  imports: [RouterOutlet, Counter],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class App {
  protected readonly title = signal('ngrx-store-demo');
}
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - Componente (ts):

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { CounterActions } from '../../counter/counter.actions';
import { AsyncPipe } from '@angular/common';

@Component({
  selector: 'app-counter',
  imports: [AsyncPipe],
  templateUrl: './counter.html',
  styleUrl: './counter.css',
})
export class Counter {
  counter

  constructor(private store: Store<{ count: number }>) {
    this.counter = this.store.select('count');
  }

  incrementar() { this.store.dispatch(CounterActions.increment()); }
  decrementar() { this.store.dispatch(CounterActions.decrement()); }
  reiniciar() { this.store.dispatch(CounterActions.reset()); }
}
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - Componente (html):

```
<h1>Contador con NgRx (Standalone)</h1>

<h2>{{ counter | async }}</h2>

<button (click)="incrementar()">+</button>
<button (click)="decrementar()">-</button>
<button (click)="reiniciar()">Reset</button>
```

ANGULAR

- **NgRx Store**. Ejemplo utilizando actions individuales.

ANGULAR

- **NgRx Store**. Ejemplo.
  - Actions:

```
import { createAction } from '@ngrx/store';

export const increment = createAction('[Counter] Increment');
export const decrement = createAction('[Counter] Decrement');
export const reset = createAction('[Counter] Reset');
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - Reducer:

```
import { createReducer, on } from '@ngrx/store';
import { decrement, increment, reset } from './counter.actions';

export const initialState = 0;

export const counterReducer = createReducer(
  initialState,
  on(increment, state => state + 1),
  on(decrement, state => state - 1),
  on(reset, state => 0)
);
```

ANGULAR

- **NgRx Store**. Ejemplo.
  - Componente (ts):

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { increment, decrement, reset } from '../../counter/counter.actions';
import { AsyncPipe } from '@angular/common';

@Component({
  selector: 'app-counter',
  imports: [AsyncPipe],
  templateUrl: './counter.html',
  styleUrl: './counter.css',
})
export class Counter {
  counter

  constructor(private store: Store<{ count: number }>) {
    this.counter = this.store.select('count');
  }

  incrementar() { this.store.dispatch(increment()); }
  decrementar() { this.store.dispatch(decrement()); }
  reiniciar() { this.store.dispatch(reset()); }
}
```

NgRx SignalStore

ANGULAR

- **NgRx SignalStore.**
  - Instalación: ng add @ngrx/signals@latest
  - Creación del store con ***signalStore:***

Estado

Estado inicial

```
import { signalStore, withState } from '@ngrx/signals';
import { Movie } from './movie'

type MovieSearchState = {
  movies: Movie[];
  isLoading: boolean;
  filter: { query: string; order: 'asc' | 'desc' };
};

const initialState: MovieSearchState = {
  movies: [],
  isLoading: false,
  filter: { query: '', order: 'asc' },
};
```

ANGULAR

- **NgRx SignalStore.**
  - Creación del store:

```
export const MovieSearchStore = signalStore(
  withState(initialState),
  withComputed((store) => ({
    moviesCount: computed(() => store.movies().length),
    directorSpielberg : computed(() =>
      store.movies().filter(movie=>movie.Director.toLowerCase().includes('spielberg'))
    )
  })),
  withMethods((store) => ({
    addMovie(movie: Movie): void {
      patchState(store, {
        movies: [...store.movies(), movie],
      });
    },
    deleteMovie(title: string): void {
      patchState(store, {
        movies: store.movies().filter((movie) => movie.Title !== title),
      });
    },
    updateMovie(updatedMovie: Movie): void {
      patchState(store, {
        movies: store.movies().map((movie) =>
          movie.Title === updatedMovie.Title ? updatedMovie : movie
        ),
      });
    }
  }))
);
```

ANGULAR

- **NgRx SignalStore.**
  - Registro***: app.config.ts***

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners, provideZoneChangeDetection, isDevMode }
from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideStoreDevtools } from '@ngrx/store-devtools';
import { MovieSearchStore } from './store/movie-search-store';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    provideStoreDevtools({ maxAge: 25, logOnly: !isDevMode() }),
    MovieSearchStore
]
};
```

ANGULAR

- **NgRx SignalStore.**
  - Uso en componentes*:*

```
readonly store = inject(MovieSearchStore);
```

```
<h1>Movie Search Store <span class="counter">{{store.moviesCount()}}</span></h1>
@for (item of store.movies(); track $index) {
    <div>{{item.Title}}</div>
}
```

ANGULAR

- **NgRx SignalStore**.
  - withState → Define la estructura de datos inicial del estado de la tienda.
  - withComputed → Define valores derivados o calculados a partir del estado u otras señales.
  - withMethods →Define las funciones de lógica de negocio para interactuar con el store.
  - withHooks → Añade métodos del ciclo de vida al store.
  - withEntities → Gestiona una colección de entidades (listas de objetos).

ANGULAR

- **NgRx SignalStore**.
  - Consulta del estado del Store:
    - npm install @angular-architects/ngrx-toolkit
  - Instalar Angular Dev Tools en Chrome.
  - En la sección Angular de las Dev Tools de Chrome, seleccionar el componente que hace uso del Store.

ANGULAR

- Enlaces:
  - RxJS:
    - https://rxjs.dev/
  - El patrón Flux:
    - https://medium.com/@edusalguero/patron-flux-475d858af405
    - https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/
  - Redux:
    - https://redux.js.org/
    - https://github.com/reduxjs/redux
  - NgRx:
    - https://ngrx.io/
    - https://medium.com/@james.dev/state-management-simplified-a-beginners-guide-to-ngrx-signalstore-with-angular-9a87efb06786
    - https://medium.com/@zansoriam/ngrx-en-angular-aprende-con-un-ejemplo-pr%C3%A1ctico-5c8da520a7e2
    - https://medium.com/@suni4596/state-management-and-how-it-can-be-handled-in-javascript-frameworks-5a5fe344375a
  - NgRx/signals:
    - https://ngrx.io/guide/signals