Reactividad básica 18/20

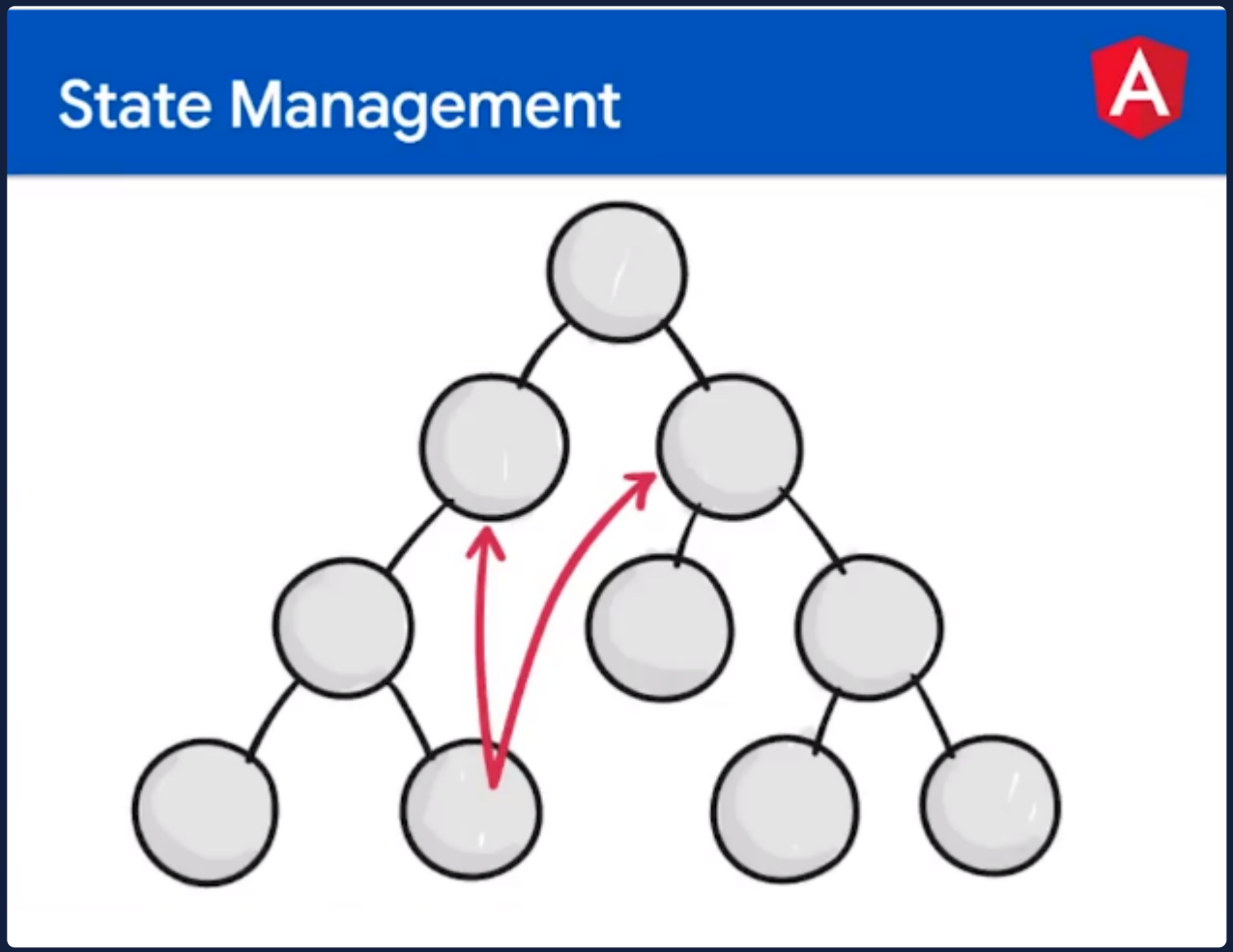
RECURSOS MARCADORES

El concepto de **reactividad básica** es muy importante en el desarrollo front-end. Se trata del **estado de la aplicación con respecto al valor de los datos** en cada componente, cómo estos cambian a medida que el usuario interactúa y cómo se actualiza la interfaz.

Problemas en la comunicación de componentes

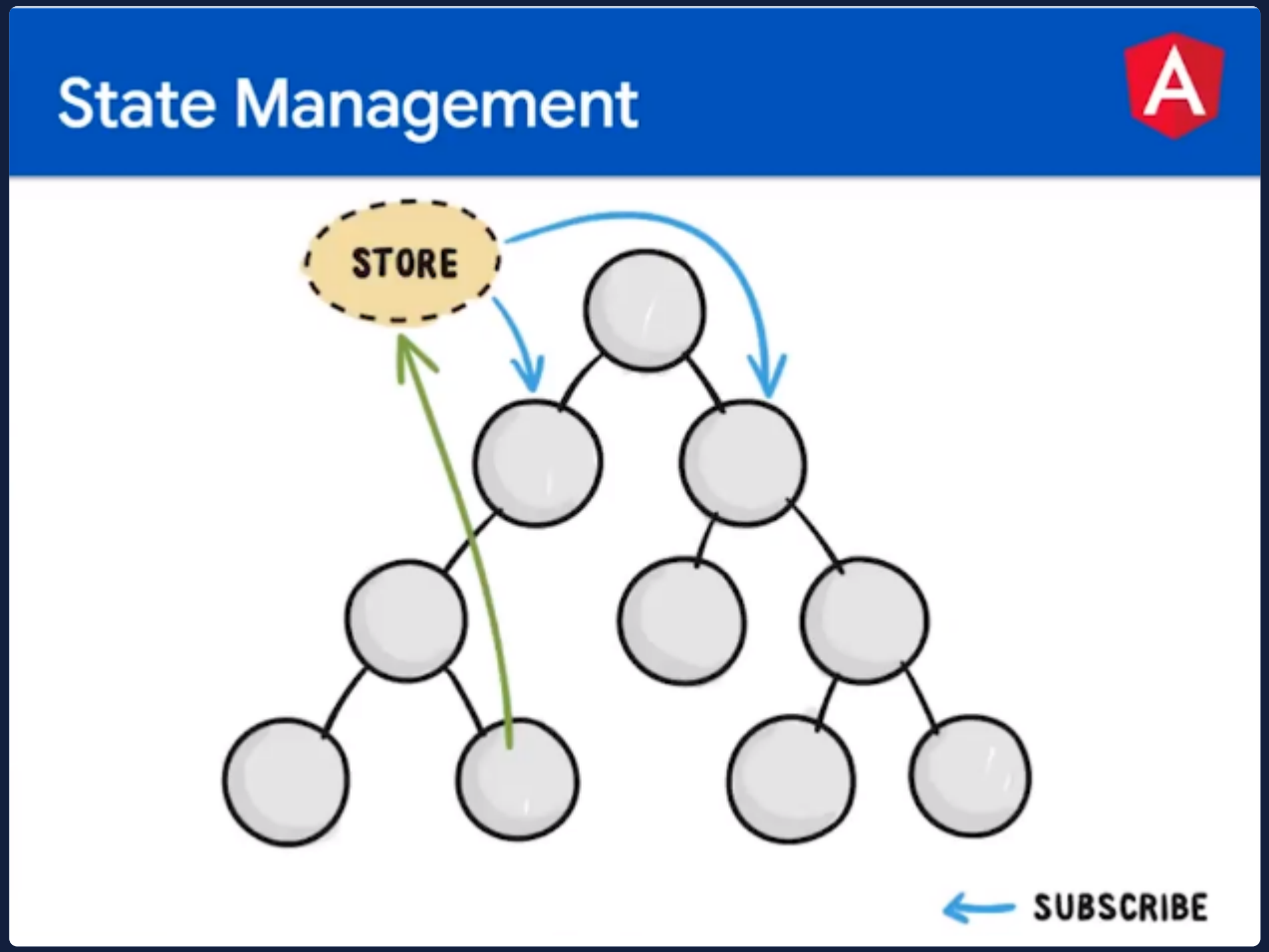
Cuando pensamos en cómo comunicar un componente padre con su hijo y viceversa, solemos utilizar los decoradores `@Input()` y `@Output()` .

Pero muchas veces, en aplicaciones grandes, la comunicación de componentes se vuelve mucho más compleja y estas herramientas no alcanzan cuando se necesita enviar información de un componente “hijo” a uno “abuelo”.



Solución a la comunicación de componentes

Es recomendable implementar un patrón de diseño para mantener el estado de la aplicación centralizado en un único punto, para que todos los componentes accedan a ellos siempre que necesiten. A este punto central se lo conoce como **Store**.



{height=""
width=""}

Implementando un *store* de datos

Los *store* de datos suelen implementarse haciendo uso de Observables.

Paso 1:
Importa la clase `BehaviorSubject` desde la librería `RxJS`, que te ayudará a crear una propiedad observable, a la cual tu componente pueda suscribirse y reaccionar ante ese cambio de estado.

```
// services/store.service.ts
import { BehaviorSubject } from 'rxjs';

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

  private myShoppingCart: Producto[] = [];
  private myCart = new BehaviorSubject<Producto[]>([]);
  public myCart$ = this.myCart.asObservable();

  constructor() { }

  addProducto(producto: Producto): void {
    // El observable emitirá un nuevo valor con cada producto que se agregue al carrito.
    this.myShoppingCart.push(producto);
    this.myCart.next(this.myShoppingCart);
  }

}
```

Paso 2: Suscribe a cualquier componente que necesites a estos datos, para reaccionar cuando estos cambian.

```
// components/nav-bar/nav-bar.component.ts
import { StoreService } from 'src/app/services/store.service';
import { Subscription } from 'rxjs';
```

```ts
@Component({
  selector: 'app-nav-bar',
  templateUrl: './nav-bar.component.html',
  styleUrls: ['./nav-bar.component.scss']
})
export class NavBarComponent implements OnInit, OnDestroy {

  private sub$!: Subscription;

  constructor(
    private storeService: StoreService
  ) { }

  ngOnInit(): void {
    this.storeService.myCart$
      .subscribe(data => {
        // Cada vez que el observable emita un valor, se ejecutará este código
        console.log(data);
      });
  }

  ngOnDestroy(): void {
    this.sub$.unsubscribe();
  }

}
```

El lugar más apropiado para esto es en `ngOnInit()`. No olvides guardar este observable en una propiedad del tipo `Subscription` para hacer un `unsubscribe()` cuando el componente sea destruido.

> *NOTA: Por convención, las propiedades que guardan observables suelen tener un **"$"** al final del nombre para indicar que se trata de un observable.*

Ver código fuente del proyecto

Contribución creada con los aportes de Kevin Fiorentino.

Lecturas recomendadas

- GitHub - platzi/angular-componentes at 15-step
  https://github.com/platzi/angular-componentes/tree/15-step
- Angular
  https://angular.io/guide/observables-in-angular