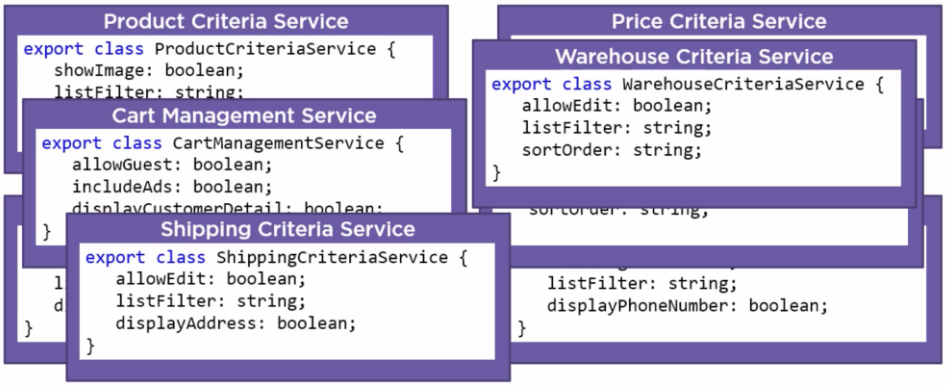
Introducción - ¿Por qué usar NgRx?

Sin *NgRx*, a medida que construimos nuestra aplicación, podemos utilizar un servicio para definir algunos estados que conservamos para nuestras páginas. A medida que vayamos añadiendo más funcionalidades a nuestra aplicación, iremos creando más y más servicios con propósitos diferentes. Con el tiempo, podemos encontrarnos con miles de estos pequeños servicios.



Con NgRx, no necesitamos un servicio para cada bit de estado. En su lugar, tenemos un único *Store* para el estado de nuestra aplicación, por lo que es fácil de encontrar, rastrear y recuperar los valores de estado.

Podemos tener un servicio de acceso a datos que se parezca al siguiente:

Product Service

```
export class ProductService {
  private productUrl = 'api/products';

  constructor(private http: HttpClient) { }

  getProducts(): Observable<Product[]> {
    return this.http.get<Product[]>(this.productUrl);
  }
}
```

En la imagen tenemos un método GET básico que carga nuestros productos desde un servicio back-end usando HTTP. Sin NgRx, nuestro componente, por ejemplo, ProductListComponent, llama a ese servicio:

Product List Component

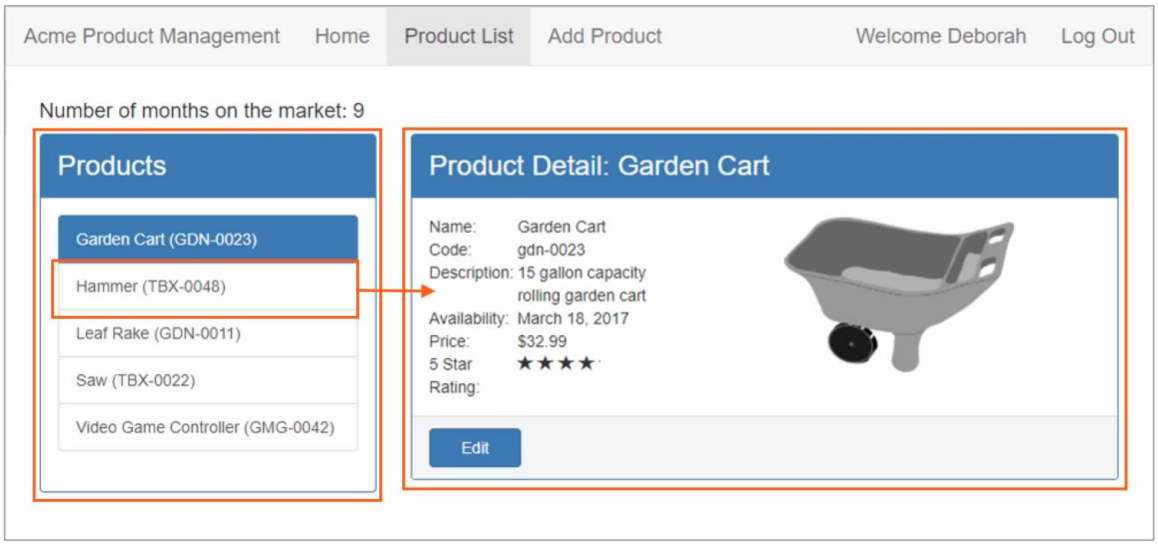
```
export class ProductListComponent {
  products: Product[] = [];

  constructor(private productService: ProductService) { }

  ngOnInit(): void {
   this.productService.getProducts()
     .subscribe(products => this.products = products);
  }
}
```

Vea que el código está dentro del método NgOnInit. Eso significa que cuando el usuario navega a esta página, el componente llama al servicio para emitir una petición HTTP GET para recuperar los productos del servidor. Si el usuario navega a otra página, aunque sea sólo un momento, y vuelve a navegar de regreso a nuestro componente, el componente llama al servicio para obtener de nuevo los datos. Si esos datos cambian con frecuencia, puede ser lo deseado, pero algunos datos simplemente no cambian tanto, por lo que no hay necesidad de obtenerlos de nuevo a medida que el usuario navega a través de la aplicación. Podemos implementar manualmente una caché de datos en nuestro servicio para evitar esta recarga constante de datos.

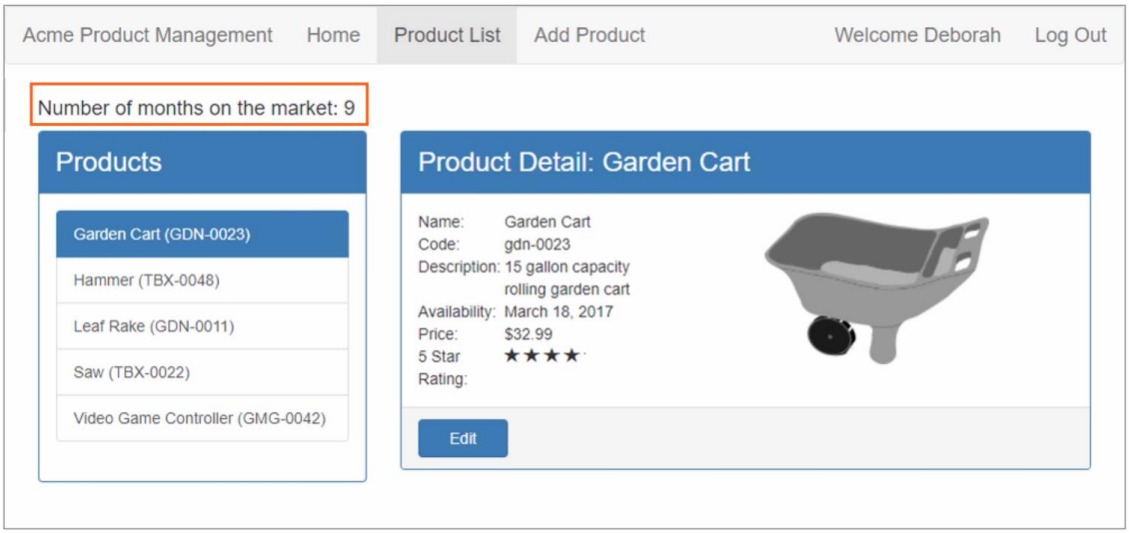
Con NgRx, *el Store proporciona la caché del lado del cliente*, por lo que no necesitamos recuperar los datos del servidor cada vez que se inicializa el componente.

Sin NgRx, tenemos que manejar explícitamente la interacción de componentes complejos.

En el siguiente ejemplo, cada vez que el usuario selecciona un nuevo producto de la lista, la aplicación notifica a la página de detalles para que pueda mostrar los detalles apropiados:



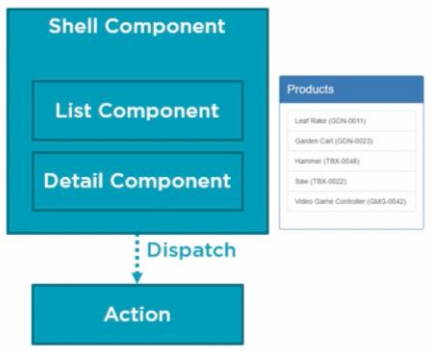
También notifica al componente padre para que pueda mostrar el número correcto de meses:



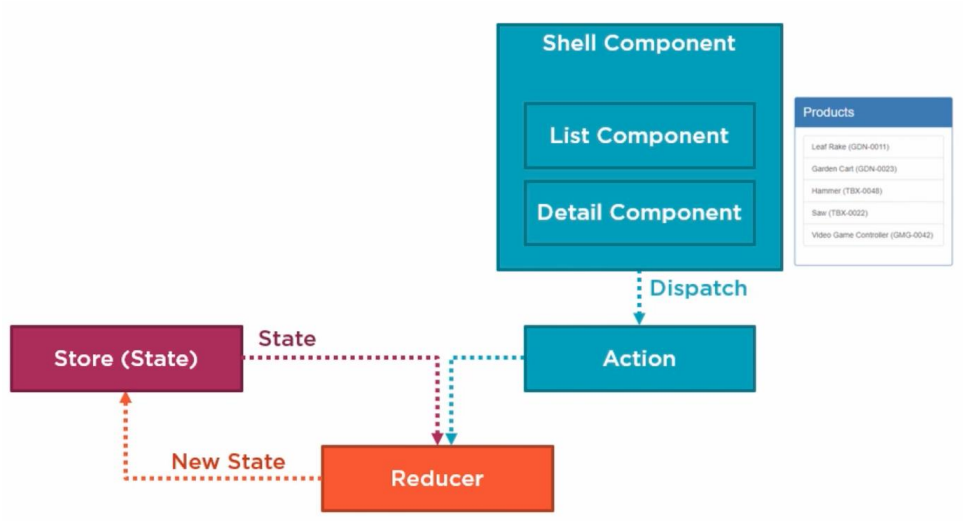
Asegurarse de que se notifica el cambio de estado a cada componente sin que haya un acoplamiento fuerte entre ellos puede suponer un poco de trabajo.

Con NgRx, cada vez que el usuario selecciona un producto:

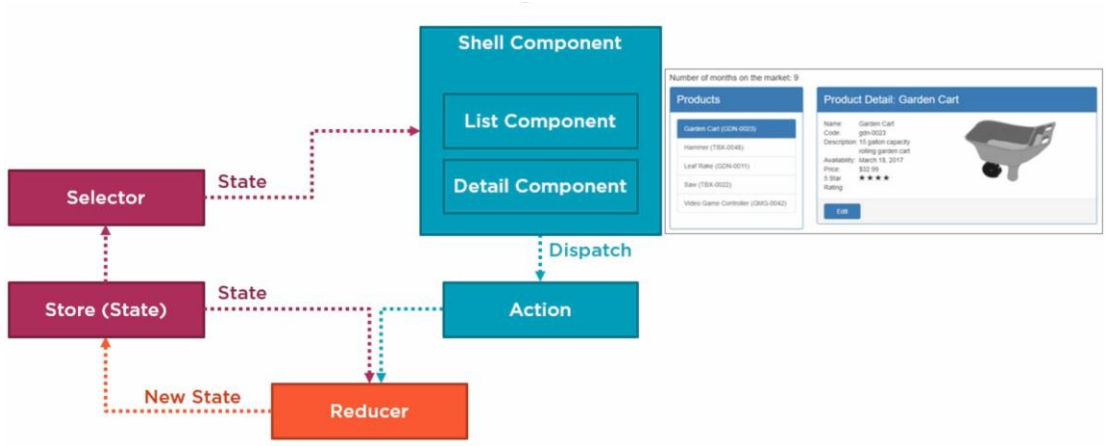
1. El componente envía un *Action*.



2. El *Reducer* utiliza ese *Action* y el *estado actual* de la aplicación desde el Store para definir un nuevo estado y actualiza el Store con ese nuevo estado. A continuación, el Store almacena el nuevo estado.



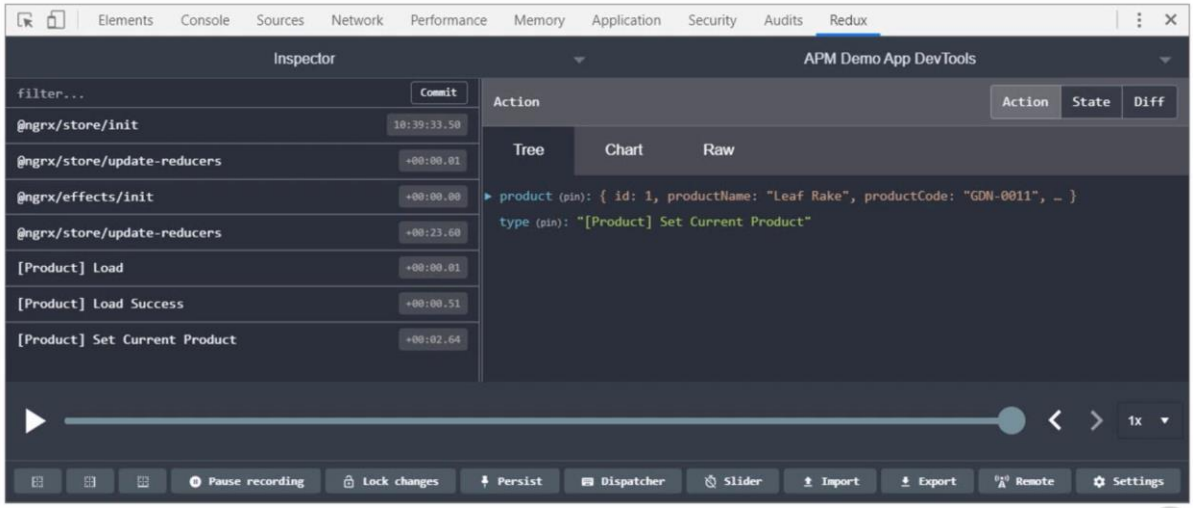
3. Cualquier componente puede suscribirse al *Selector* del producto seleccionado para recibir notificaciones de cambio, y sus vistas asociadas muestran la información apropiada para el producto actual.



Todo está perfectamente *desacoplado*:

- **Los componentes no introducen datos directamente en el Store:** En lugar de ello, envían un Action para que el Reducer lo actualice.
- **Los componentes no leen datos directamente del Store:** En su lugar, se suscriben al Store a través de un Selector para recibir notificaciones de cambios de estado.
- Todos los componentes están recibiendo el estado de una única fuente de la verdad.

El estado almacenado en el Store se presenta de forma coherente. Si algo no está funcionando correctamente, *NgRx tiene herramientas* que nos permiten ver nuestra lista de Actions y nuestro estado, haciendo mucho más fácil ver lo que está pasando en nuestra aplicación.



Poniendo todo junto, es bueno utilizar NgRx cuando:

- **Hay muchos estados en la aplicación:** El Store proporciona un lugar para el estado de la UI para poder retenerlo cuando se navega entre vistas.
- **Hay muchas peticiones HTTP:** El Store proporciona una caché del lado del cliente para utilizar según se requiera.
- **Hay interacciones complejas entre componentes:** El Reducer actualiza el Store, y el Store notifica a todos los suscriptores, manteniendo los componentes desacoplados, pero comunicados. El uso de NGRx en este escenario puede evitar condiciones raras y problemas causados cuando múltiples componentes están actualizando datos.
- **Necesite un patrón estándar y confiable que se pueda depurar:** Cuando usamos NgRx y algo no está funcionando, tiene grandes herramientas para ayudarnos a ver nuestros Actions y nuestro estado.

Pero NgRx no es para todos los proyectos.

No utilice NgRx cuando:

- **Usted y su equipo son nuevos con Angular:** Aprenda primero con los Observables de RxJs. Tendrá mucho más éxito con NgRx cuando domine RxJs.
- **Su aplicación es pequeña y sencilla:** Puede que no necesite NgRx si su aplicación no maneja muchos estados. El código extra requerido para implementar NgRx puede no merecer la pena para una aplicación simple.
- **Ya implementa una buena forma de manejar estados:** Si su equipo ya tiene un buen patrón de gestión de estados para sus aplicaciones, entonces puede no merecer la pena cambiar a NgRx, dependiendo de las ventajas que le da el patrón que ya implementa