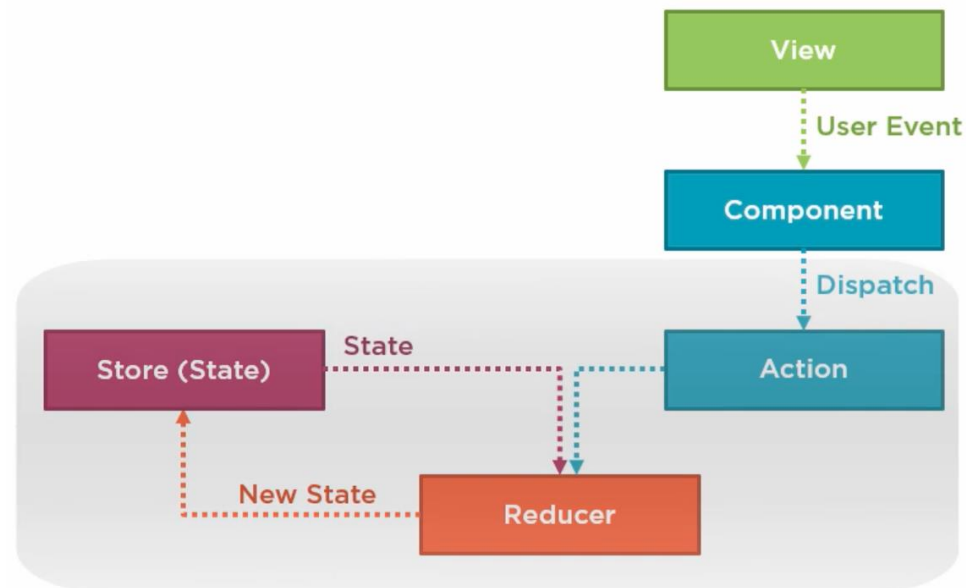
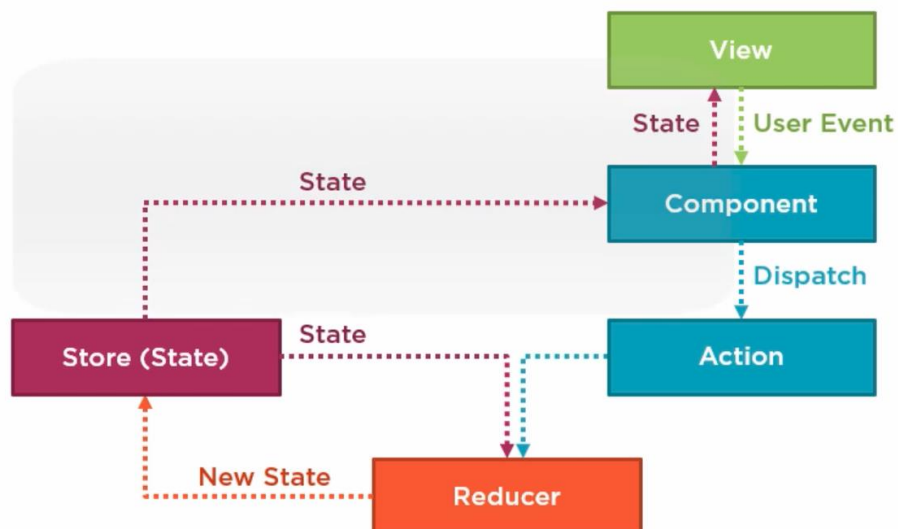


## Primer vistazo a NgRx – Suscribirse al Store para obtener los cambios de estado

Cada vez que el usuario hace clic en el checkbox, se emite el Action se ejecuta el Reducer y el estado existente del Store se sustituye por el nuevo estado, incluido el valor del checkbox:



Pero nos falta la pieza final, aún no estamos obteniendo el valor del Store:



Por eso nuestra UI no refleja el cambio de estado.

Para acceder a un valor del Store, seleccionamos el slice de estado adecuado. La selección se realiza mediante el método “select” del Store:

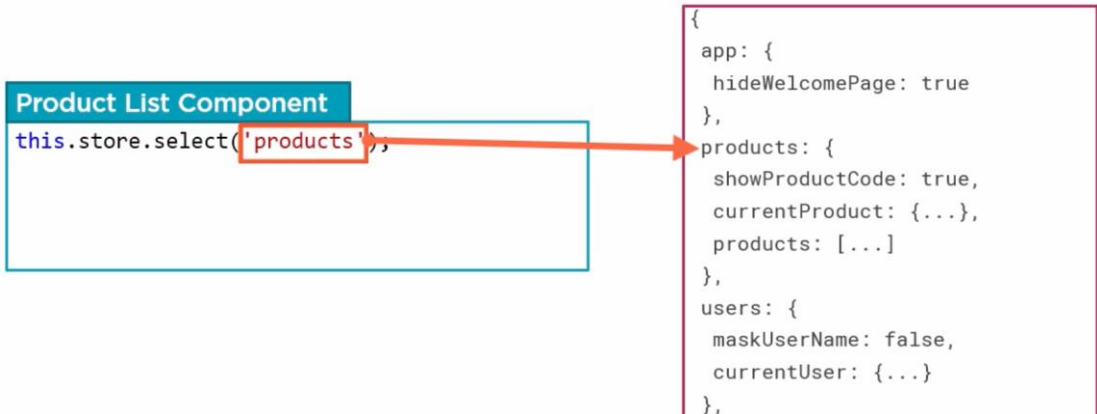
#### Product List Component

```
this.store.select('products');
```

Especificamos el nombre del slice de estado deseado, que en este ejemplo es nuestro slice “products”:

#### Product List Component

```
this.store.select('products');
```



```
{
  app: {
    hideWelcomePage: true
  },
  products: {
    showProductCode: true,
    currentProduct: {...},
    products: [...]
  },
  users: {
    maskUserName: false,
    currentUser: {...}
  },
}
```

An orange arrow points from the 'products' string in the code block to the 'products' property in the state object.

El método NgRx “select” devuelve un slice de estado como un Observable. Si queremos que nuestro componente sea notificado de cambios de estado, nos suscribimos a este Observable:

#### Product List Component

```
this.store.select('products').subscribe(
  products =>
    this.displayCode = products.showProductCode
);
```

Note que ahora estamos suscritos a este slice completo de estado:

#### Product List Component

```
this.store.select('products').subscribe(  
  products =>  
    this.displayCode = products.showProductCode  
);
```

```
{  
  app: {  
    hideWelcomePage: true  
  },  
  products: {  
    showProductCode: true,  
    currentProduct: {...},  
    products: [...]  
  },  
  users: {  
    maskUserName: false,  
    currentUser: {...}  
  },  
}
```

Así que cualquier cambio en cualquier bit de este estado en el slice “products” genera una notificación y proporciona todo el slice de estado.

El primer argumento del método “subscribe” es una función next:

#### Product List Component

```
this.store.select('products').subscribe(  
  products =>  
    this.displayCode = products.showProductCode  
);
```

Esta función se ejecuta cada vez que recibe la siguiente notificación de cambio del Store. En esta función, establecemos nuestra propiedad local “displayCode” al bit de estado deseado de nuestro slice de estado:

#### Product List Component

```
this.store.select('products').subscribe(  
  products =>  
    this.displayCode = products.showProductCode  
);
```

```
{  
  app: {  
    hideWelcomePage: true  
  },  
  products: {  
    showProductCode: true,  
    currentProduct: {...},  
    products: [...]  
  },  
  users: {  
    maskUserName: false,  
    currentUser: {...}  
  },  
}
```

Así, cada vez que un Action hace que el Reducer reemplace el estado, si el slice “products” fue afectado, el componente es notificado y este código se ejecuta.

Ahora vamos a completar nuestro primer flujo de datos NgRx y a suscribirnos a los cambios de estado.


Si nos fijamos en el componente `product-list.component`, queremos recibir notificaciones cada vez que cambie el estado de los productos. Vamos a añadir el código para suscribirse a los cambios en el método `ngOnInit`, por lo que inmediatamente proporciona los valores existentes del Store:



```
28  ngOnInit(): void {
29    this.sub = this.productService.selectedProductChanges$.subscribe({
30      selectedProduct => this.selectedProduct = selectedProduct
31    });
32
33    this.productService.getProducts().subscribe({
34      next: (products: Product[]) => this.products = products,
35      error: err => this.errorMessage = err
36    });
37
38    this.store.select('products').subscribe(
39      products => {
40        if (products) {
41          this.displayCode = products.showProductCode;
42        }
43      }
44    );
45
46    ngOnDestroy(): void {
47      this.sub.unsubscribe();
48    }
49  }
```

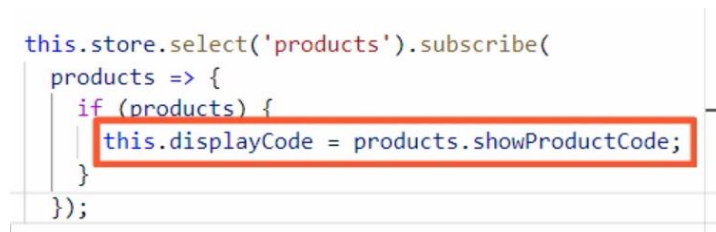
Aquí utilizamos el método `select`. El argumento es el nombre del *slice* de estado. Luego nos suscribimos para recibir notificaciones de cambio de estado. Cuando nos suscribimos, y cada vez que el estado cambia, recibimos todo el *slice* `products`.

Cuando la aplicación se ejecuta por primera vez, su *slice* de estado está definida por el estado inicial que establecimos en el *Reducer*, pero el *slice* de estado “products” puede no estar definida todavía, así que agregamos una validación para esperar a que se reciba una notificación con el *slice* de estado existente:



```
this.store.select('products').subscribe(
  products => {
    if (products) {
      this.displayCode = products.showProductCode;
    }
  }
);
```

A continuación, establecemos nuestra propiedad local con el valor de nuestro *slice* de estado:



```
this.store.select('products').subscribe(
  products => {
    if (products) {
      this.displayCode = products.showProductCode;
    }
  }
);
```

Nuestra plantilla se vincula a esta propiedad local y la utiliza en un ngIf para alternar la visualización de los códigos de producto:

```
TS product-list.component.ts X ... product-list.component.html X TS product.reducer.ts
28 ngOnInit(): void {
29   this.sub = this.productService.selectedProductChanges$;
30   selectedProduct => this.selectedProduct = selectedPr
31 );
32
33   this.productService.getProducts().subscribe({
34     next: (products: Product[]) => this.products = produ
35     error: err => this.errorMessage = err
36   });
37
38   this.store.select('products').subscribe(
39     products => {
40       if (products) {
41         this.displayCode = products.showProductCode;
42       }
43     });
44 }
45
46 ngOnDestroy(): void {
47   this.sub.unsubscribe();
48 }
49
50 checkChanged(): void {
51   this.store.dispatch(
52     { type: 'checkChanged', payload: this.selectedProduct }
53   );
54 }
55 }

7 <div class="card-body"
8   *ngIf="products?.length"
9 <div class="list-group"
10 <button class="list-group-item list-gr
11   *ngFor="let product of products"
12   [ngClass]="{'active': product?.
13   (click)="productSelected(product)"
14   <ng-container *ngIf="displayCode">
15     {{{ product.productCode }}}
16   </ng-container>
17 </button>
18 </div>
19 </div>
20
21 <div class="card-footer">
22 <div class="row">
23 <div class="form-check col-md-7">
24 <label>
25 <input class="form-check-input"
26   type="checkbox"
27   (change)="checkChanged()"
28   [checked]="displayCode">
29   Display Product Code
30 </label>
31 </div>
32 </div>
33 </div>
```

Si se ha suscrito a observables antes sabe que también tenemos que darnos de baja de ellos. Una forma de hacerlo es almacenar la suscripción en una variable, como lo hacemos aquí:

```
TS product-list.component.ts X ...
28 ngOnInit(): void {
29   this.sub = this.productService.selectedProductChanges$;
30   selectedProduct => this.selectedProduct = selectedPr
31 );
32
33   this.productService.getProducts().subscribe({
34     next: (products: Product[]) => this.products = produ
35     error: err => this.errorMessage = err
36   });
37
38   this.store.select('products').subscribe(
39     products => {
40       if (products) {
41         this.displayCode = products.showProductCode;
42       }
43     });
44 }
```

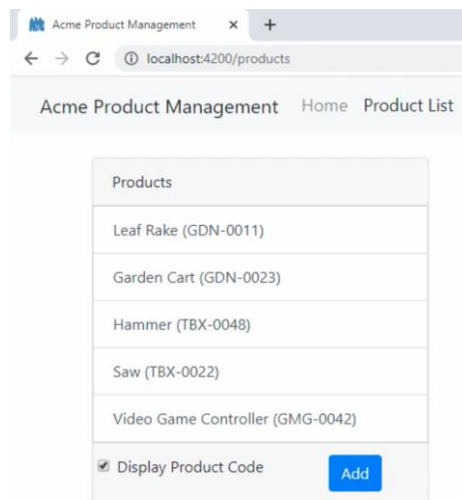
Después usamos esa variable para darnos de baja, como lo hacemos aquí:

```
37
38   this.store.select('products').subscribe(
39     products => {
40       if (products) {
41         this.displayCode = products.showProductCode;
42       }
43     });
44
45
46   ngOnDestroy(): void {
47     this.sub.unsubscribe();
48   }
49
50   checkChanged(): void {
51     this.store.dispatch(
52       { type: '[Product] Toggle Product Code' }
53     );
54   }
55 }
```

Pero a medida que añadimos más suscripciones a un componente, almacenar cada una en una variable separada se volverá más complejo. Así que dejaremos para más adelante las estrategias más adecuadas para darnos de baja de las suscripciones.

Con esto nos hemos suscrito a los cambios del Store.

Al abrir el navegador, vemos que el checkbox “Display Product Code” está marcado desde el inicio y también aparecen los códigos de los productos:



Nuestro componente *product-list.component* lee el estado inicial del Store.

Si desmarcamos el checkbox, los códigos desaparecen. Marcamos de nuevo el Checkbox, navegamos hacia el “Home Page”, volvemos a la página “Product List” y nuestro Store conserva nuestro estado y el checkbox se reestablecerá al valor establecido anteriormente.

Con esto hemos completado nuestra primera implementación de un flujo básico con NgRx.