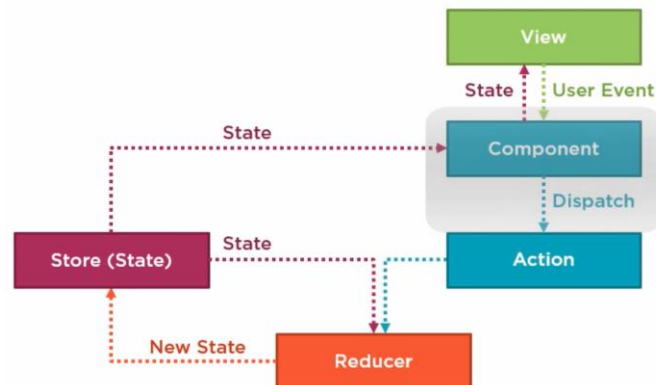


Primer vistazo a NgRx – Emitir un Action para cambiar el estado (Demo)

Ya tenemos nuestro Store inicializado, nuestro Estado y nuestro Reducer implementados, peor no pasa nada hasta que emitamos un Action.

A menudo emitimos Actions desde nuestros componentes basados en eventos del usuario:



Antes de poder enviar un Action desde un componente, debemos inyectar el Store en ese componente. Al igual que cualquier otro servicio, inyectamos el servicio del Store utilizando el constructor:

```
Product List Component  
constructor(private store: Store<any>) {}
```

Nótese el tipo de dato any con el que declaramos la inyección del Store. Aún no estamos tipando fuertemente nuestro estado, por lo que el argumento de tipo genérico debe ser any.

Cuando el usuario realiza una operación, como marcar el checkbox “Display Product Code”, emitimos un Action:

Products	
Leaf Rake	
Garden Cart	
Hammer	
Saw	
Video Game Controller	
<input checked="" type="checkbox"/> Display Product Code	<button>Add</button>

Emitimos el Action llamando al método dispatch del Store y pasando el Action:

Product List Component

```
constructor(private store: Store<any>) {}

checkChanged(): void {
  this.store.dispatch({
    type: '[Product] Toggle Product Code'
  });
}
```

Por ahora, definiremos nuestro Action como un objeto literal, con una propiedad type establecida con el mismo string que definimos en el Reducer:

Product List Component

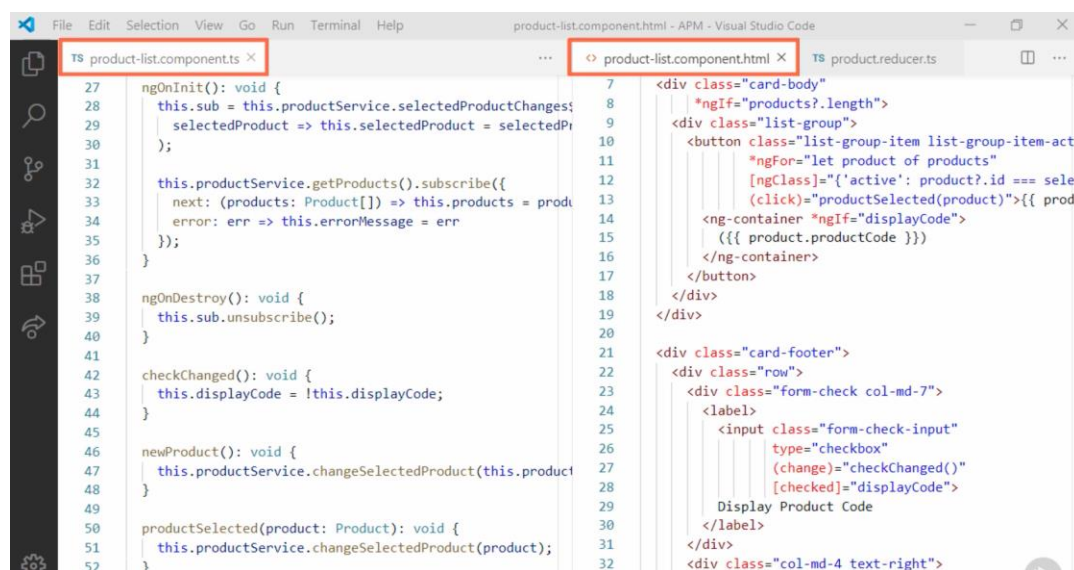
```
constructor(private store: Store<any>) {}

checkChanged(): void {
  this.store.dispatch({
    type: '[Product] Toggle Product Code'
  });
}
```

Hacer coincidir strings de esta forma es propenso a errores, pero nuestro objetivo en este punto es un simple primer vistazo a un flujo con NgRx. Veremos más adelante como tipear fuertemente nuestros Actions y asignar metadatos a esos Actions.

En esta demo, enviaremos nuestro Action “toggleProductCode” cada vez que el usuario haga clic en el checkbox “Display Product Code”.

Tenemos nuestro archivo product-list.component.ts y su plantilla:



El checkbox “Display Product Code” está definido en la plantilla **product-list.component.html**:

```
<div class="card-body">
  *ngIf="products?.length">
  <div class="list-group">
    <button class="list-group-item list-group-item-action"
      *ngFor="let product of products"
      [ngClass]="{'active': product?.id === selectedProduct?.id}"
      (click)="productSelected(product)">{{ product.name }}
    <ng-container *ngIf="displayCode">
      {{ product.productCode }}
    </ng-container>
  </button>
</div>
</div>

<div class="card-footer">
  <div class="row">
    <div class="form-check col-md-7">
      <label>
        <input class="form-check-input"
          type="checkbox"
          (change)="checkChanged()"
          [checked]="displayCode">
        Display Product Code
      </label>
    </div>
  </div>
</div>
```

Ya tenemos el código para utilizar event binding y llamar al método “checkChanged” cada vez que se marca el checkbox:

```
product-list.component.ts
27 ngOnInit(): void {
28   this.sub = this.productService.selectedProductChanges$.subscribe(
29     selectedProduct => this.selectedProduct = selectedProduct
30   );
31
32   this.productService.getProducts().subscribe({
33     next: (products: Product[]) => this.products = products
34     error: err => this.errorMessage = err
35   });
36 }
37
38 ngOnDestroy(): void {
39   this.sub.unsubscribe();
40 }
41
42 checkChanged(): void {
43   this.displayCode = !this.displayCode;
44 }
45
46 newProduct(): void {
47   this.productService.changeSelectedProduct(this.selectedProduct);
48 }
49

product-list.component.html
7 <div class="card-body">
8   *ngIf="products?.length">
9   <div class="list-group">
10    <button class="list-group-item list-group-item-action"
11      *ngFor="let product of products"
12      [ngClass]="{'active': product?.id === selectedProduct?.id}"
13      (click)="productSelected(product)">{{ product.name }}
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>
```

Mirando el archivo *product-list.component.ts*, el método “checkChange” actualmente alterna una propiedad local llamada “displayCode” usando el operador “not” de JavaScript:

```
37
38   ngOnDestroy(): void {
39     this.sub.unsubscribe();
40   }
41
42   checkChanged(): void {
43     this.displayCode = !this.displayCode;
44   }
45
46   newProduct(): void {
47     this.productService.changeSelectedProduct(this.produ
48   }
```

En la plantilla, esa propiedad local se utiliza con una directiva *ngIf* para ocultar o mostrar el código del producto en la lista:

```
<> product-list.component.html X TS product.reducer.ts
7   <div class="card-body"
8     *ngIf="products?.length">
9     <div class="list-group">
10      <button class="list-group-item list-group-item-act
11        *ngFor="let product of products"
12        [ngClass]='{"active": product?.id === sele
13        (click)="productSelected(product)">{{ prod
14        <ng-container *ngIf="displayCode">
15          ({{ product.productCode }})
16        </ng-container>
17      </button>
18    </div>
19  </div>
```

Esto funciona bien hasta que el usuario sale de la página de la lista de productos. Como la propiedad “displayCode” es local a este componente, su valor se pierde cuando se destruye el componente:

```
TS product-list.component.ts X ...
13   export class ProductListComponent implements OnInit, OnDe
14     pageTitle = 'Products';
15     errorMessage: string;
16
17     displayCode: boolean;
18
19     products: Product[];
20
21     // Used to highlight the selected product in the list
22     selectedProduct: Product | null;
23     sub: Subscription;
24
25     constructor(private productService: ProductService) { }
26
```

Por lo tanto, si el usuario vuelve a esta página, el componente no recordará la configuración anterior.

Ahora emitamos un Action para retener el valor en el Store.

En el componente, primero inyectamos el Store en el constructor y definimos el tipo de argumento genérico como any, e importamos la librería respectiva:

```
TS product-list.component.ts X
13 })
14 export class ProductListComponent implements OnInit, OnDestroy {
15     pageTitle = 'Products';
16     errorMessage: string;
17
18     displayCode: boolean;
19
20     products: Product[];
21
22     // Used to highlight the selected product in the list
23     selectedProduct: Product | null;
24     sub: Subscription;
25
26     constructor(private store: Store<any>, private productService: ProductService) {}
27
28     ngOnInit(): void {
29         this.sub = this.productService.selectedProductChange
30             .subscribe(selectedProduct => this.selectedProduct = selectedProduct);
31     }
32 }
```

A continuación, sustituimos el código del método “checkChange” por código para emitir el Action. Llamamos al método “dispatch” del Store inyectado y le pasamos nuestro Action:

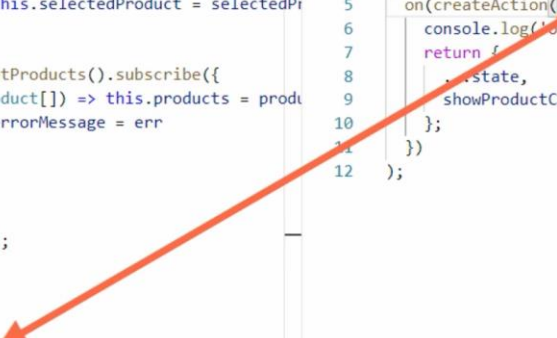
```
27
28     ngOnInit(): void {
29         this.sub = this.productService.selectedProductChange
30             .subscribe(selectedProduct => this.selectedProduct = selectedProduct);
31     };
32
33     this.productService.getProducts().subscribe({
34         next: (products: Product[]) => this.products = products,
35         error: err => this.errorMessage = err
36     });
37 }
38
39     ngOnDestroy(): void {
40         this.sub.unsubscribe();
41     }
42
43     checkChanged(): void {
44         this.store.dispatch(action);
45     }
```


¿Pero cómo especificamos aquí nuestro Action cuando todavía no usamos tipado fuerte? Si nos fijamos en el Reducer, hemos definido el Action con un nombre string:

```
<> product-list.component.html TS product.reducer.ts X
1  import { createReducer, on, createAction } from '@ngrx/
2
3  export const productReducer = createReducer(
4    { showProductCode: true },
5    on(createAction('[Product] Toggle Product Code'), st
6      console.log('original state: ' + JSON.stringify(st
7      return {
8        ...state,
9        showProductCode: !state.showProductCode
10     };
11   });
12 );
```

Para emitir este Action, pasamos un objeto Action con una propiedad type establecida a ese string:

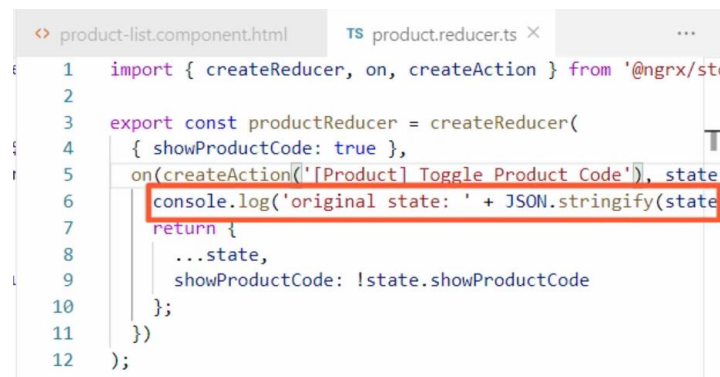
```
TS product-list.component.ts X TS product.reducer.ts X
26 constructor(private store: Store<any>, private productService: ProductService) {}
27
28 ngOnInit(): void {
29   this.sub = this.productService.selectedProductChanges$.subscribe(
30     selectedProduct => this.selectedProduct = selectedProduct
31   );
32
33   this.productService.getProducts().subscribe({
34     next: (products: Product[]) => this.products = products,
35     error: err => this.errorMessage = err
36   });
37 }
38
39 ngOnDestroy(): void {
40   this.sub.unsubscribe();
41 }
42
43 checkChanged(): void {
44   this.store.dispatch(
45     { type: '[Product] Toggle Product Code' }
46   );
47 }
```



Este Action se emitirá cada vez que el usuario haga clic en el checkbox "Display Product Code".

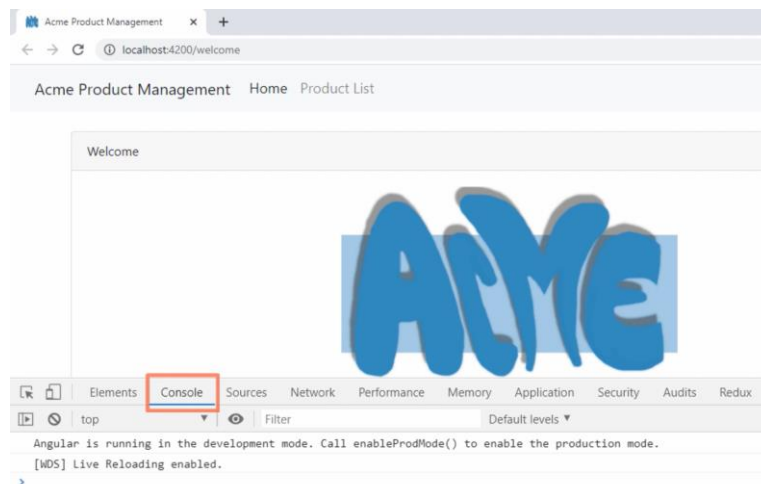
Vamos a probarlo.

Recordemos que añadimos código en nuestro Reducer para imprimir el estado antes de modificarlo, y establecimos el valor inicial de la propiedad “showPRoductCode” a true:

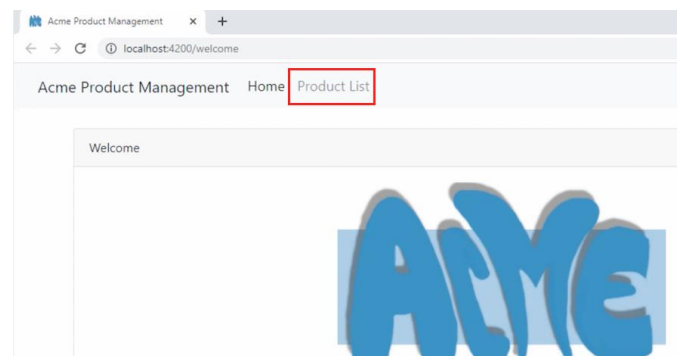


```
1 import { createReducer, on, createAction } from '@ngrx/store';
2
3 export const productReducer = createReducer(
4   { showProductCode: true },
5   on(createAction('[Product] Toggle Product Code'), state => {
6     console.log('original state: ' + JSON.stringify(state));
7     return {
8       ...state,
9       showProductCode: !state.showProductCode
10    };
11  });
12 );
```

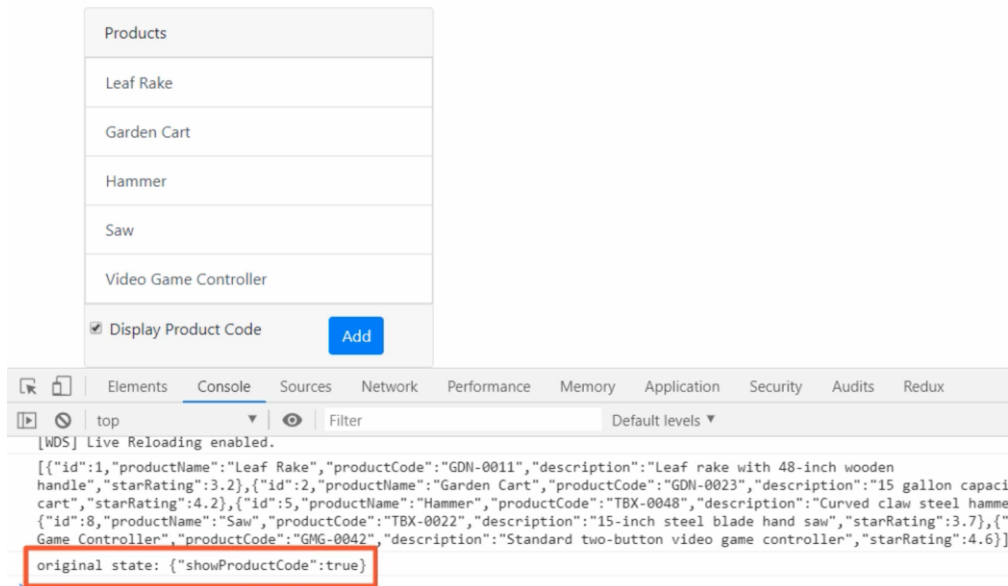
En el navegador, abra las herramientas de desarrollo con la pestaña “Console” seleccionada para que podamos ver el log del Reducer:



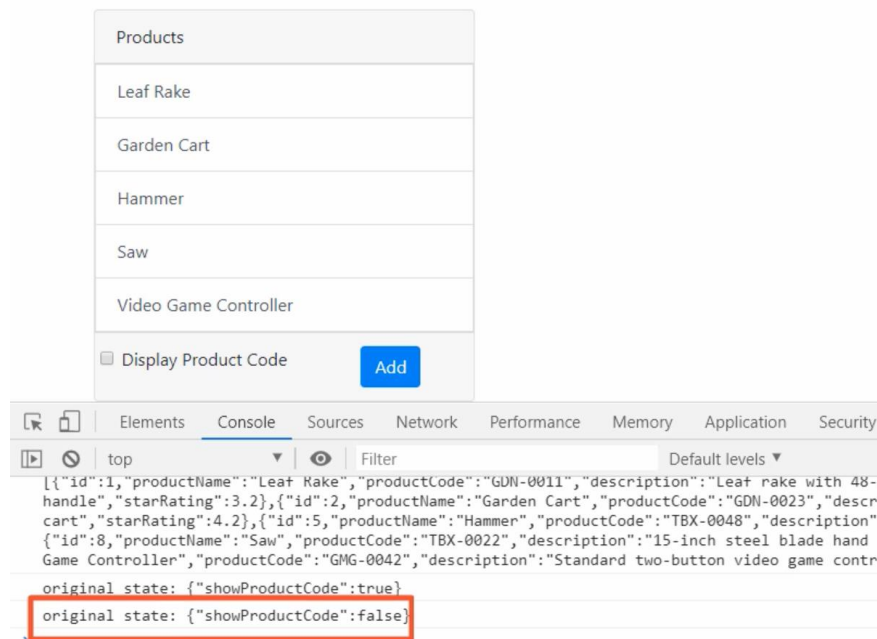
A continuación, haz clic en la opción del menú superior “Product List”:



A continuación, haga clic en el checkbox “Display Product Code”. Vemos que el estado original tiene la propiedad “showPRoductCode” establecida a true:

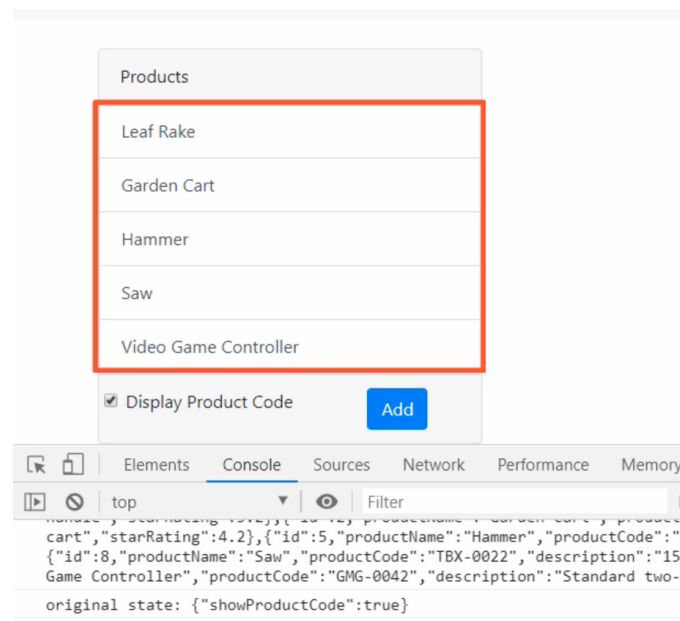


¿Se dónde ha salido ese valor? Vino del estado inicial que establecimos en el Reducer. A continuación, haga clic en el heckbox de nuevo, y vea como se actualiza el estado:



Parece que nuestro Reducer funciona correctamente.

Pero observe que al hacer clic en el checkbox no se muestran los códigos de productos:



¿Por qué no aparecen los códigos de los productos? Eso es porque todavía no estamos recuperando el estado del Store.

Ahora, vamos a completar nuestro flujo y obtener los cambios de estado de nuestro Store.