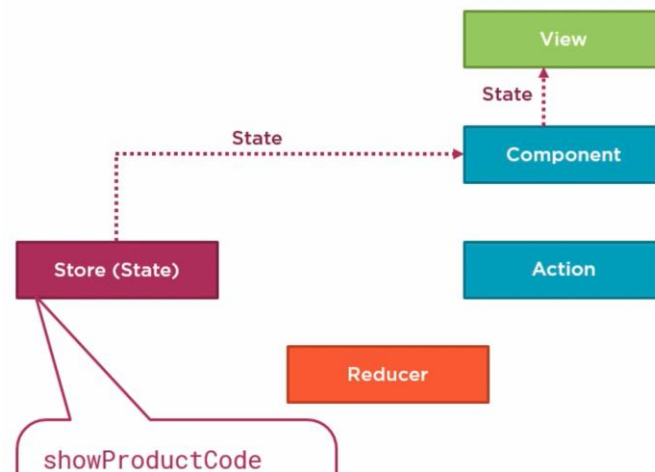


## Tipado fuerte del estado – Establecer los valores iniciales del estado

Cuando un componente se suscribe por primera vez al *Store*, obtiene el valor actual de su *slice* de estado solicitado. Si el estado aún no se ha modificado, el componente obtiene el valor inicial del estado establecido en el Reducer. Actualmente sólo hemos inicializado el flag *showProductCode*.



Dado que uno de los objetivos de utilizar NgRx es hacer que nuestra aplicación sea más predecible, debemos definir explícitamente los valores iniciales para cada bit de estado.


Para inicializar nuestro estado, definimos un objeto y establecemos un valor inicial para cada bit de estado. Para asegurar que el estado inicial nunca cambie, lo declaramos como constante.

### Product Reducer

```
const initialState: ProductState = {  
  showProductCode: true,  
  currentProduct: null,  
  products: []  
};
```

Aquí definimos una constante llamada *initialState*, y especificamos un valor inicial para cada propiedad en el *slice* de estado “*product*”. La constante la creamos utilizando nuestra interfaz *ProductState*. El tipeado fuerte de *initialState* garantiza que se establezca cada propiedad del estado con un valor.

Una vez definida la constante, la pasamos como primer argumento en la función *createReducer*:



```
Product Reducer
const initialState: ProductState = {
  showProductCode: true,
  currentProduct: null,
  products: []
};

Product Reducer
export const productReducer = createReducer<ProductState>(
  initialState,
  on(ProductActions.toggleProductCode, (state): ProductState =>
  {
    return {
      ...state,
      showProductCode: !state.showProductCode
    };
  })
);
```

The diagram illustrates the flow of the initial state. A red box at the top defines `initialState`. A red arrow points from this box to a second red box below it, which shows `initialState` being passed as the first argument to `createReducer` in the `productReducer` function.

Ahora, cuando se inicializa el *Store* y se llama al *Reducer* por primera vez, se asignan estos valores iniciales y nuestras propiedades del *Store* nunca estarán indefinidas.

Vamos a aplicar esta configuración inicial.

Dado que, tanto las interfaces como el *Reducer* están en el archivo *product.reducer.ts*, tiene sentido inicializar el *State* aquí también:



```
TS product.reducer.ts X TS product-list.component.ts
5
6 export interface State extends AppState.State {
7   products: ProductState;
8 }
9
10 export interface ProductState {
11   showProductCode: boolean;
12   currentProduct: Product;
13   products: Product[];
14 }
15
16 export const productReducer = createReducer<ProductState>(
17   { showProductCode: true } as ProductState,
18   on(createAction('[Product] Toggle Product Code'), (state): ProductState => {
19     return {
20       ...state,
21       showProductCode: !state.showProductCode
22     };
23   })
24 );
```

The screenshot shows a code editor with two tabs: `TS product.reducer.ts` (active) and `TS product-list.component.ts`. The active tab contains the following code:

Inmediatamente después de la interfaz, definiremos nuestro *initialState*:

```
TS product.reducer.ts X TS product-list.component.ts
5
6 export interface State extends AppState.State {
7   products: ProductState;
8 }
9
10 export interface ProductState {
11   showProductCode: boolean;
12   currentProduct: Product;
13   products: Product[];
14 }
15
16 const initialState: ProductState = {
17   showProductCode: true,
18   currentProduct: null,
19   products: []
20 };
21
22 export const productReducer = createReducer<ProductState>(
23   { showProductCode: true } as ProductState,
24   on(createAction('[Product] Toggle Product Code'), (state): ProductState => {
25     return {
26       ...state,
27       showProductCode: !state.showProductCode
28     };
29   })
30 );
```

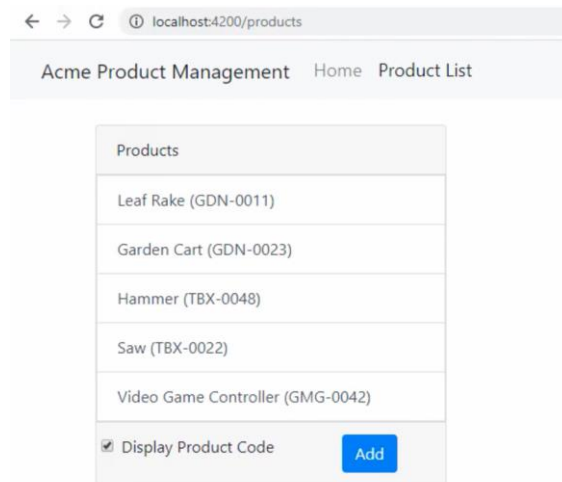
Declaramos *initialState* como una constante, le damos un nombre y establecemos su tipo con nuestro tipo de interfaz.

A continuación, modificamos el primer argumento pasado a la función *createReducer* para pasarle la constante *initialState*:

```
22 export const productReducer = createReducer<ProductState>(
23   { showProductCode: true } as ProductState,
24   on(createAction('[Product] Toggle Product Code'), (state): ProductState => {
25     return {
26       ...state,
27       showProductCode: !state.showProductCode
28     };
29   })
30 );

22 export const productReducer = createReducer<ProductState>([
23   initialState,
24   on(createAction('[Product] Toggle Product Code'), (state): ProductState => {
25     return {
26       ...state,
27       showProductCode: !state.showProductCode
28     };
29   })
30 ]);
```

Volvemos a nuestro navegador y podemos ver que nuestro valor por default de *initialState* inicializa el heckbox con true.



Ahora, mirando el archivo product-list.component, ya no necesitamos comprobar nuestro estado antes de acceder a él, porque ahora está predefinido:

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

Podemos borrar esta condición *if* y simplificar nuestra función de flecha:

```
39   // TODO: Unsubscribe
40   this.store.select('products').subscribe(
41     products => this.displayCode = products.showProductCode;
42   );
43 }
```

A pesar de que hemos tipado fuertemente e inicializado el estado, todavía tenemos un string en código duro en el *select* del *slice* de “products”, pero podemos hacerlo mejor:

```
TS product.reducer.ts TS product-list.component.ts X
26
27 constructor(private store: Store<State>, private productService: ProductService) { }
28
29 ngOnInit(): void {
30   this.sub = this.productService.selectedProductChanges$.subscribe(
31     selectedProduct => this.selectedProduct = selectedProduct
32   );
33
34   this.productService.getProducts().subscribe({
35     next: (products: Product[]) => this.products = products,
36     error: err => this.errorMessage = err
37   });
38
39   // TODO: Unsubscribe
40   this.store.select('products').subscribe(
41     products => this.displayCode = products.showProductCode
42   );
43 }
44
```