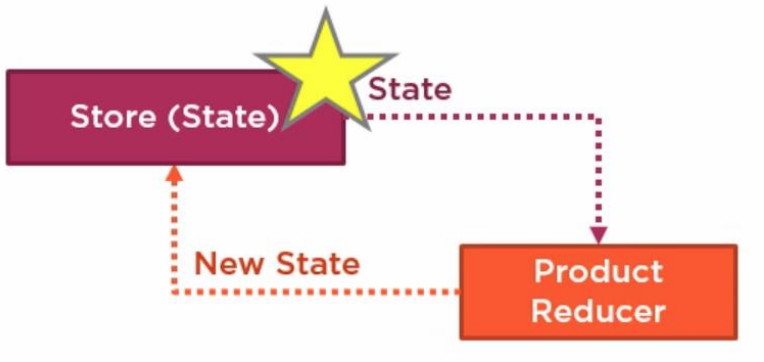
Primer vistazo a NgRx – Construir un Reducer para procesar los Actions

Hemos definido nuestro estado y nuestro Action. Ahora estamos listos para construir un Reducer para procesar ese Action y crear un nuevo estado.

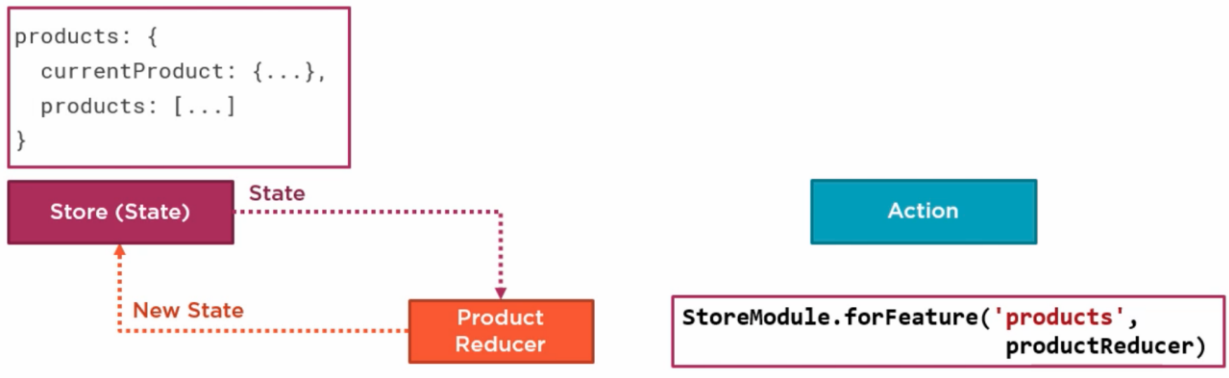
Aunque el Store es la estrella del espectáculo de NgRx, es el Reducer el que hace todo el trabajo.



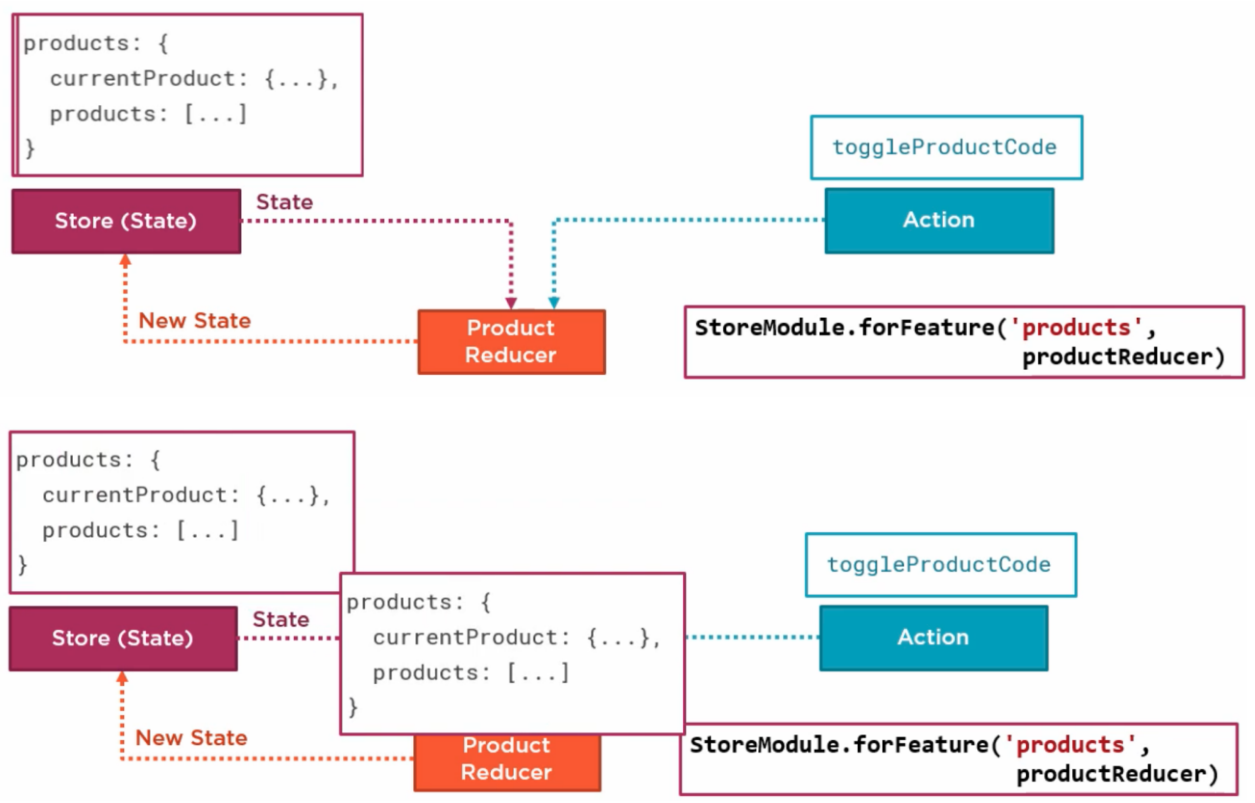
A medida que los Actions se emiten, es el Reducer el que realiza cualquier procesamiento necesario y devuelve una **nueva representación del estado del Store**. Este es es un concepto clave de NgRx. El árbol de estado del Store debe ser inmutable, lo que significa que su estructura y valores nunca deben cambiar. Espera, ¿qué? Si se supone que no debemos cambiar el estado del Store, ¿cómo procesamos nuestros Actions? Veamos un ejemplo:

1. Registramos el Reducer de nuestro producto en el módulo de feature products cuando inicializamos el Store, por lo que el Reducer de nuestro producto representa el *slice* "products" de nuestro Store:

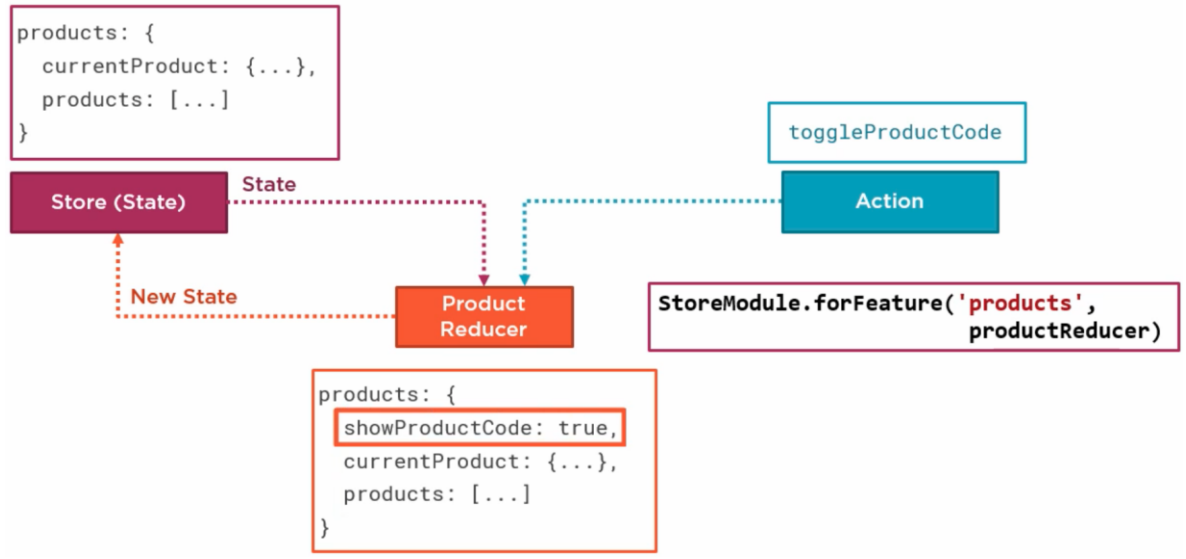
2. Digamos que el *slice* "products" incluye los datos del producto actualmente seleccionado y la lista de productos para mostrar en la lista de productos:



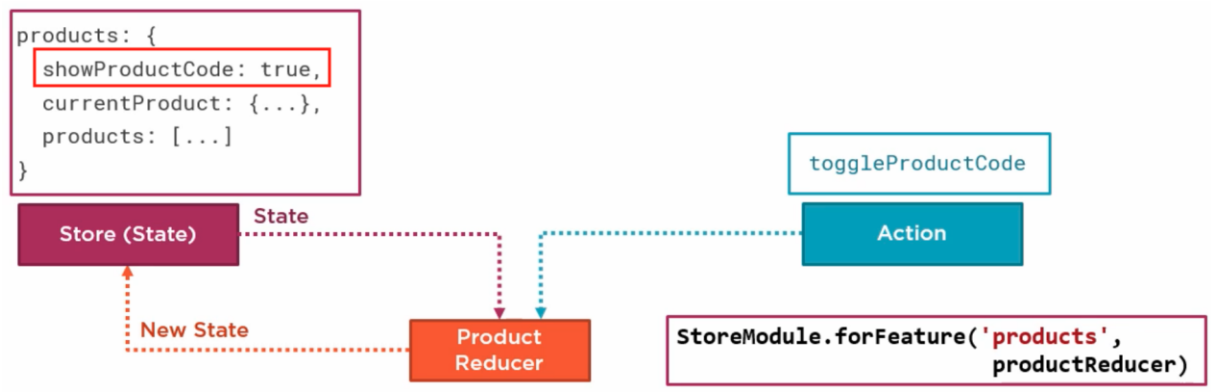
3. El usuario hace clic en el checkbox "Display Product Code" y se emite el Action "toggleProductCode". El Reducer toma ese Action y el *slice* asociado al Reducer, que en este caso es todo el *slice* "products":



4. A continuación, el Reducer crea un nuevo estado copiando el estado existente y aplicando cambios a esa copia basándose en el Action emitido, en este ejemplo, añadiendo la variable showProductCode con un valor de true:



5. El Reducer sustituye el *slice* del árbol de estado correspondiente a “products” por este nuevo estado, creando así un nuevo estado:



Así que, técnicamente, no estamos modificando ni mutando el estado.

¿Por qué tanto trabajo para conseguir la inmutabilidad? La inmutabilidad hace que los cambios de estado sean más explícitos y predecibles, lo que es importante a medida que la aplicación se hace más grande o más compleja. También evita que nos hagamos preguntas como ¿en dónde demonios se está cambiando esto? Y todos los cambios son definidos por Actions específicos que pueden ser seguidos y rastreados.

Veamos ahora cómo luce un Reducer.

Product Reducer

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

Un Reducer se especifica con una constante que se exporta, definiendo una función Reducer usando la función *createReducer*. A menudo, creamos un Reducer para cada *feature slice* de estado, por lo que nombramos la constante con el nombre del *feature slice*:

Product Reducer

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

La función *createReducer* toma un estado inicial, que define los valores iniciales para el *slice* "product":

Product Reducer

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

El propósito de la función Reducer es escuchar los Actions, y para cada Action, manejar la transición del estado actual al nuevo estado de una manera inmutable. Identificamos esos Actions y sus hadlers usando una o más funciones llamadas "on":

Product Reducer

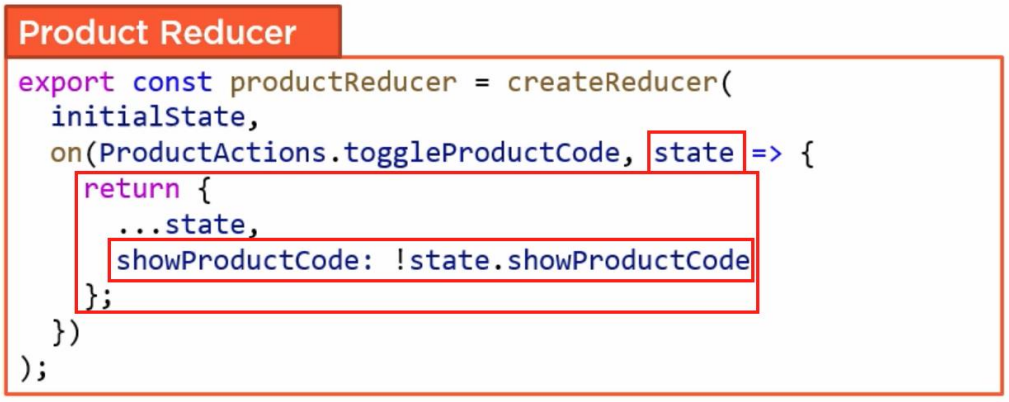
```
export const productReducer = createReducer(
  initialState,
  on(ProductActions.toggleProductCode, state => {
    return {
      ...state,
      showProductCode: !state.showProductCode
    };
  })
);
```

La función "on" toma el Action, y una función manejadora ese Action:

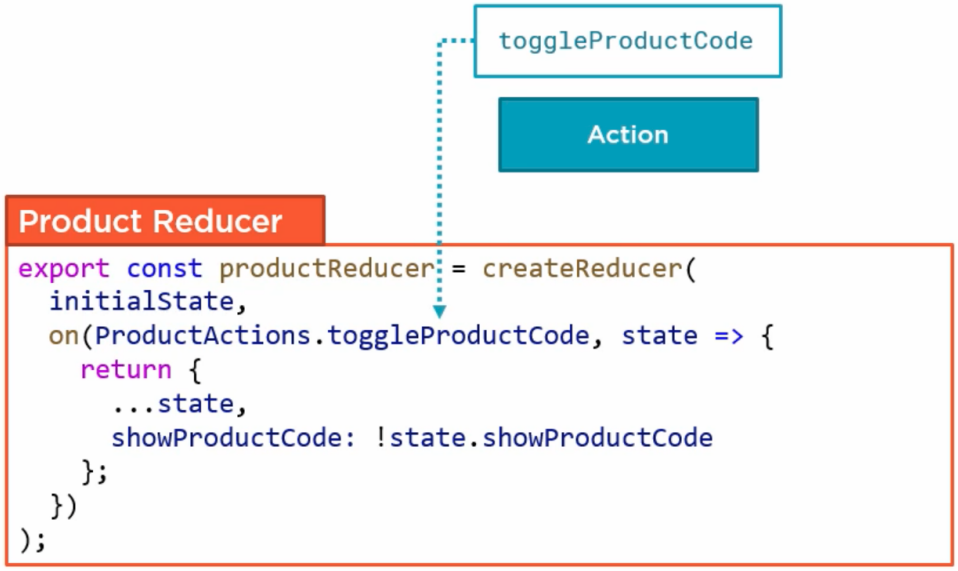
Product Reducer

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

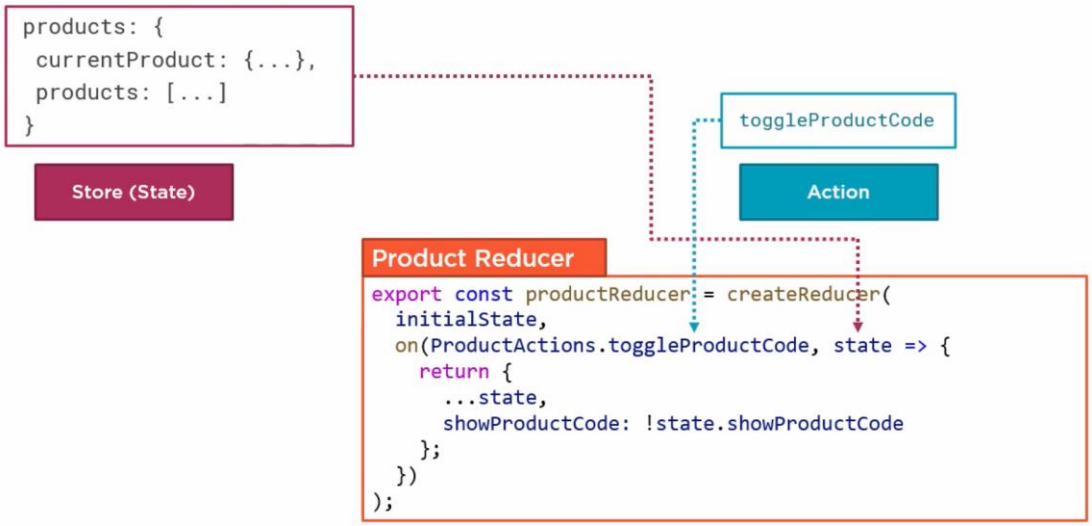
Esta función toma el estado actual, lo procesa según sea necesario y devuelve el nuevo estado:



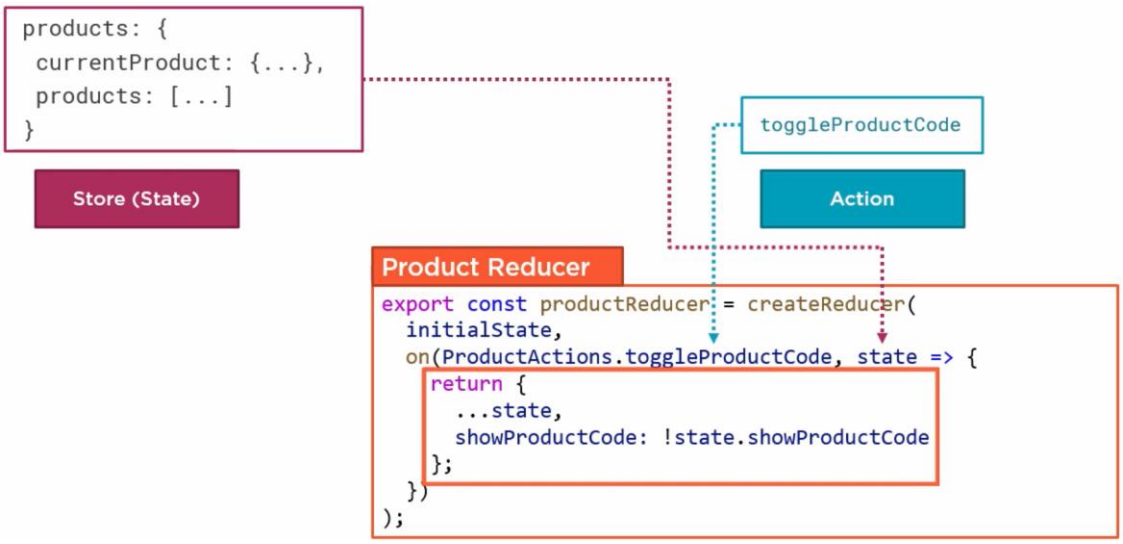
En este ejemplo, escuchamos el Action “toggleProductCode”:



Cuando se emite ese Action, se pasa el estado actual al Reducer:



Cuando se recibe el estado actual, se ejecuta la *función handler* y se retorna el nuevo estado:



Una cosa importante a tener en cuenta aquí es que el Reducer debe ser una función pura. Es decir, que, dada la misma entrada, la función siempre devuelve la misma salida sin tener "side effects". Además, cada transición de estado debe ser sincrónica.

En este ejemplo, sólo tenemos definido un Action. Si elegimos utilizar dos Actions para nuestro escenario, nuestro Reducer tendría dos funciones "on":

Product Reducer

```
export const productReducer = createReducer (
  initialState,
  on(ProductActions.showProductCode, state => {
    return {
      ...state,
      showProductCode: true
    };
  }),
  on(ProductActions.hideProductCode, state => {
    return {
      ...state,
      showProductCode: false
    };
  })
);
```

A medida que añadimos más Actions para nuestros productos, añadimos más funciones "on".

También podemos especificar múltiples Actions que establezcan el mismo estado, lo que nos permite reducir la cantidad de código repetido en nuestros Reducers:

Product Reducer

```
export const productReducer = createReducer (
  initialState,
  on(ProductActions.showProductCode, state => {
    return {
      ...state,
      showProductCode: true
    };
  }),
  on(ProductActions.hideProductCode,
     ProductActions.resetDefaults, state => {
    return {
      ...state,
      showProductCode: false
    };
  })
);
```