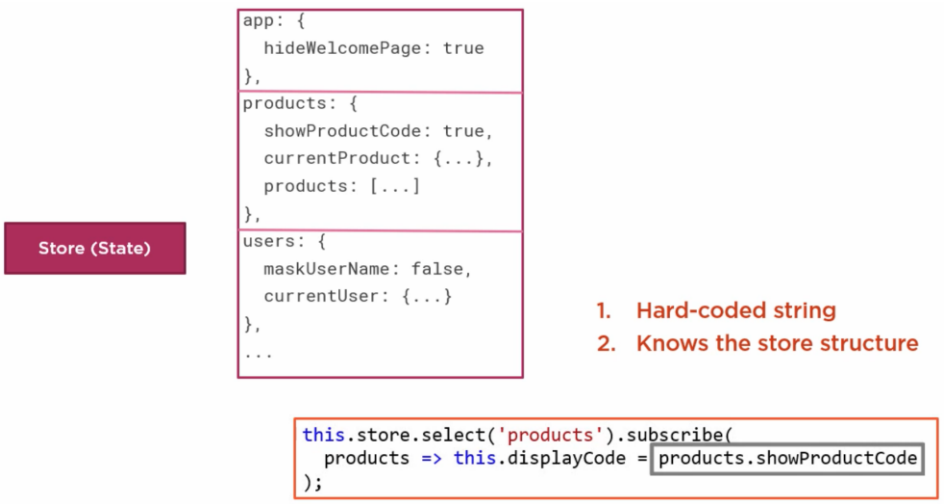
Tipado fuere del estado – Construyendo Selectores

Hasta ahora, nuestro componente se ha suscrito al Store, seleccionando todo el *slice* de estado de "product":

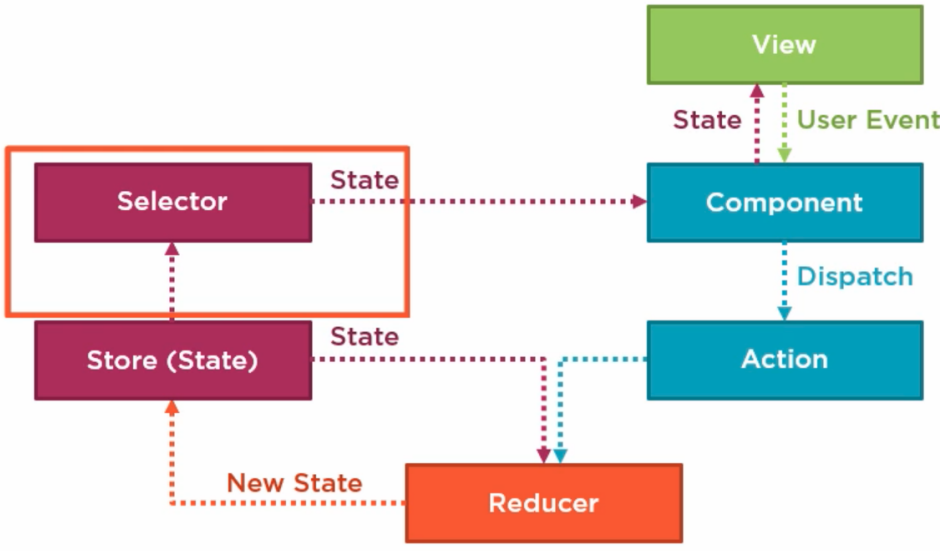
Hay algunos problemas con este enfoque. En primer lugar, tenemos un string en hard-code que nos deja expuestos a errores tipográficos y de escritura:

En segundo lugar, recuperamos explícitamente una propiedad del Store, haciendo suposiciones sobre la estructura del Store, lo que significa que, si alguna vez cambiamos la estructura del Store, reorganizándolo en *sub slices*, por ejemplo, tenemos que encontrar cada *select* y actualizar su código:



Por último, este código está atento a los cambios de cualquier propiedad en el *slice* de estado de "product", por lo que este código notifica incluso si la propiedad "showProductCode" no ha cambiado:

¿Cómo mejorarlo? Con *Selectors*. Un Selector es un query reutilizable de nuestro Store:



Un *Selector* es básicamente como un *Stored Procedure* para acceder a nuestra información de estado en memoria. Los *Selectors* nos permiten mantener una copia del estado en el Store, pero proyectarlo en diferentes formas, facilitando el acceso a nuestros componentes y servicios. Nuestros componentes utilizan el *Selector* para seleccionar el estado de nuestro Store, añadiendo un nivel de abstracción entre nuestra estructura del Store y nuestro componente.

El uso de Selectors tiene varias ventajas:

- Proporciona una API fuertemente tipada para que la utilicen los componentes. No tenemos que referirnos a *slices* de estado con strings en hard-code.
- Desacoplan el Store de los componentes, por lo que los componentes no necesitan saber acerca de la estructura del Store. Esto nos permite reorganizar o dividir el estado de forma diferente a lo largo del tiempo sin actualizar cada componente que accede a él.
- Los *Selectors* pueden encapsular transformaciones de datos, facilitando a los componentes la obtención de los datos complejos.
- Son reutilizables, por lo que cualquier componente puede acceder al mismo bit de estado de la misma manera.
- Los *Selectors* tienen memoria, lo que significa que el valor devuelto por el *Selector* se almacena caché, y no se reevaluará a menos que cambie el estado. Esto puede mejorar el rendimiento.

Entonces ¿qué es exactamente un Selector? Es una función que profundiza en el Store y devuelve una parte específica del estado. Hay dos tipos básicos de función Selector proporcionados por la librería NgRx:

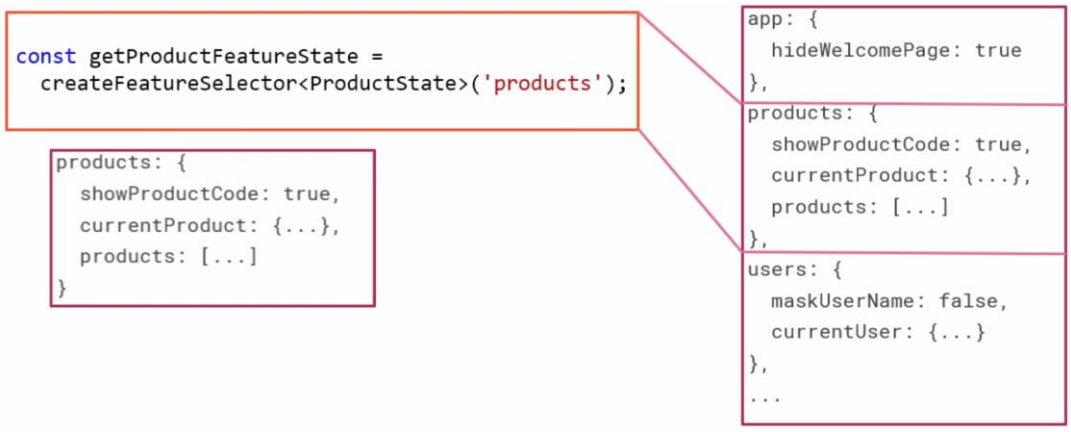
- El primer tipo es un *createFunctionSelector*:

```
const getProductFeatureState =
  createFeatureSelector<ProductState>('products');
```

  Esta función nos permite obtener el slice de estado de un feature simplemente especificando el nombre de ese feature. Tipeamos fuertemente el valor de retorno utilizando el argumento genérico:

```
const getProductFeatureState =
  createFeatureSelector<ProductState>('products');
```

Cuando se ejecuta, selecciona el slice de estado del feature especificado:



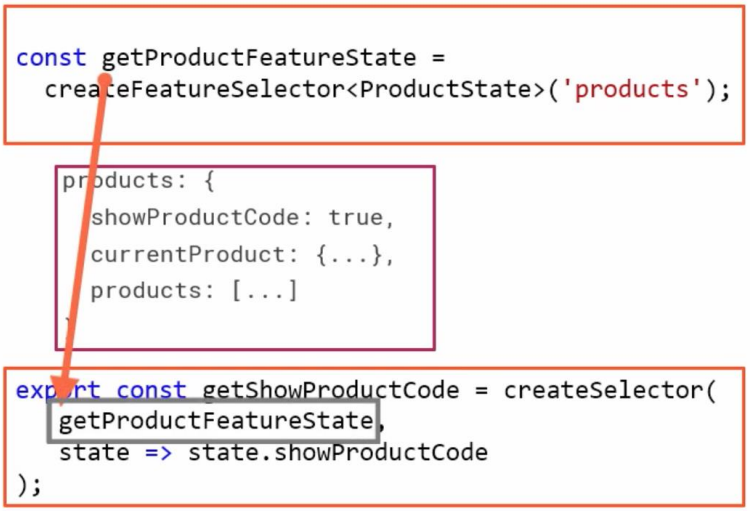
No exportamos esta constante, por lo que sólo puede utilizarse donde está definida.

- El segundo tipo de *Selector* es un *createSelector*:

```
export const getShowProductCode = createSelector(
   getProductFeatureState,
   state => state.showProductCode
);
```

Esta función nos permite obtener cualquier bit de estado componiendo Selectors para navegar por el árbol de estados.

Pasamos la función *featureSelector* como primer argumento a este *Selector*:



El último argumento es una función proyectora que toma el estado de los argumentos anteriores, que en este caso es el *slice* "product":

```
export const getShowProductCode = createSelector(
   getProductFeatureState,
   state => state.showProductCode
);
```

A continuación, podemos filtrar, mapear o procesar de cualquier otro modo el estado para devolver el valor deseado. En este caso, tomamos la propiedad *showProductCode* para devolver su valor:

```
export const getShowProductCode = createSelector(
   getProductFeatureState,
   state => state.showProductCode
);
```

Asignamos esta función a una constante exportada para poder utilizar el Selector desde nuestros componentes:

```
export const getShowProductCode = createSelector(
   getProductFeatureState,
   state => state.showProductCode
);
```

Cuando se ejecuta, selecciona solo un bit de estado específico y devuelve su valor:

Al usar *Selectors* en nuestros componentes, si la estructura de nuestro Store cambia alguna vez, podemos modificar estos *Selectors* para acceder a esas nuevas estructuras sin cambiar ninguno de los componentes que los usan. Una cosa importante a tener en cuenta es que un *Selector* debe ser una función pura, es decir, dada la misma entrada, la función siempre debe devolver la misma salida sin efectos secundarios.

Usamos el *Selector* en nuestros componentes con el operador *select* o el método *select*. Actualmente estamos usando esta técnica en nuestro *product-list.component* para seleccionar el flag *showProductCode*:

Without a selector

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

El argumento de la función *select* es un string que representa el nombre del *slice* de estado que queremos obtener:

Without a selector

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

El *select* devuelve los datos del Store asociado con este nombre. En este caso, devuelve todo el *slice* de estado de "products". Luego, navegamos manualmente hacia abajo desde el *slice* "products" hasta la propiedad deseada, por lo que nuestro código debe saber cómo navegar por el árbol de estado:

Without a selector

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

Para usar un Selector en su lugar, reemplazamos el string que está en hard-code con una referencia a la función de estado:

With a selector

```
import { State, getShowProductCode } from '../state/product.reducer';

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

Este código no sabe nada sobre la estructura de nuestro Store.