Tipado fuere del estado – Selectores Compuestos

Ya hemos visto como componer *Selectors* a partir de otros *Selectors*:

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

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

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

Cada *Selector* que creamos con *createSelector* se compone a partir del *createFeatureSelector*. Pero no estamos limitados a un solo Selector compuesto. Podemos componer varios *Selectors* para construir un único *Selector*. Veamos un ejemplo:

Digamos que en lugar de almacenar el *currentProduct* en el Store, retenemos solo el ID del producto actual:

```
app: {
  hideWelcomePage: true
},
products: {
  showProductCode: true,
  currentProductId: 5,
  products: [...]
},
users: {
  maskUserName: false,
  currentUser: {...}
},
...
```

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

El Selector para obtener esta propiedad *currentProductId* sería como el siguiente:

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

export const getCurrentProductId = createSelector(
  getProductFeatureState,
  state => state.currentProductId
);
```

Pero aún con este nuevo Selector, nuestro componente aún necesita los datos del producto actual, no solo el ID.

Para lograr esto, debemos componer aún más nuestro *Selector* del producto actual:

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

```
export const getCurrentProductId = createSelector(
  getProductFeatureState,
  state => state.currentProductId
);
```

```
export const getCurrentProduct = createSelector(
  getProductFeatureState,
  getCurrentProductId,
  (state, currentProductId) =>
    state.products.find(p => p.id === currentProductId)
);
```

Aquí estamos pasando tanto el *Selector getProductFeatureState* como el *Selector getCurrentProductId*. El resultado de cada Selector se proporciona a la función proyectora en el orden en que están definidos:

```
export const getCurrentProduct = createSelector(
  getProductFeatureState,
  getCurrentProductId,
  (state, currentProductId) =>
    state.products.find(p => p.id === currentProductId)
);
```

Así, el primer argumento aquí es el estado devuelto por el *Selector getProductFeatureState*. El segundo argumento es el ID del producto actual devuelto por el *Selector getCurrentProductId*.

Nuestra función proyectora puede entonces utilizar ambos argumentos para devolver el producto correcto:

```
export const getCurrentProduct = createSelector(
  getProductFeatureState,
  getCurrentProductId,
  (state, currentProductId) =>
    state.products.find(p => p.id === currentProductId)
);
```

Este código usa el método *find* del array products para encontrar el producto deseado utilizando el ID.

Pero ¿por qué usar la composición con dos Selectors y no simplemente hacer referencia al ID del estado como se muestra en el siguiente ejemplo?

```
export const getCurrentProduct = createSelector(
  getProductFeatureState,
  (state) =>
    state.products.find(p => p.id === state.currentProductId)
);
```

Esto tiene que ver con la encapsulación. Al definir por separado un *Selector* para cada bit de estado y luego componer los *Selectors*, la estructura del Store se puede cambiar con el tiempo con un impacto mínimo en el código. En pocas palabras, al crear *Selectors*, defina uno para cada bit de estado al que se acceda desde el Store y compóngalos según lo necesiten sus componentes y servicios.