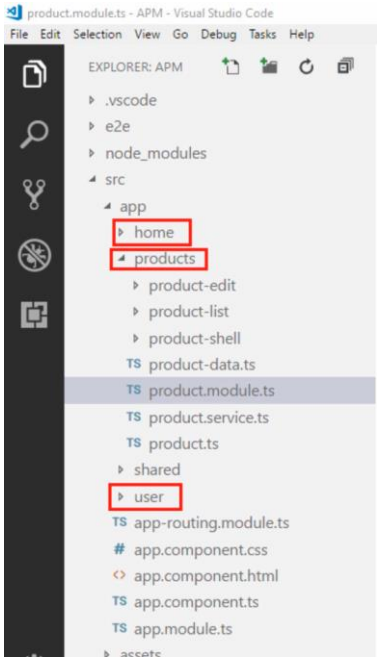
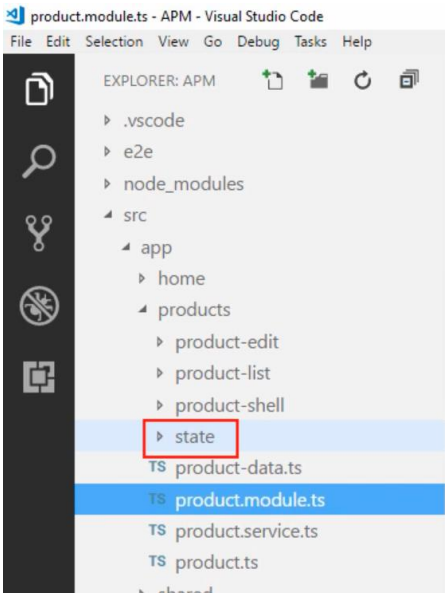
Primer vistazo a NgRx – Construir un Reducer para procesar los Actions (Demo)

En esta demo, construiremos un Reducer para procesar nuestro Action y conectaremos nuestro Reducer al Store.

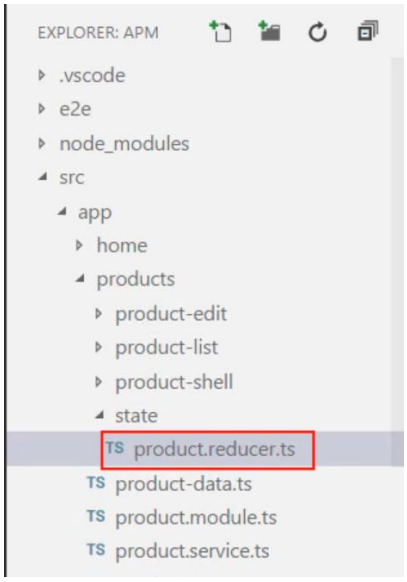
La primera pregunta que nos hacemos es ¿dónde debemos poner este Reducer? Hay varias formas de distribuir las carpetas de una aplicación. Actualmente, las carpetas de nuestra aplicación están divididas por features. Tenemos una carpeta home, una carpeta products y otra de users:



Organizaremos nuestro Estado, Actions y Reducers por features también. Así que vamos a definir una carpeta dentro de nuestra carpeta de feature products para la información del estado del feature. Llamaremos a la carpeta "state", y contendrá los Actions y Reducers para este feature:



Dentro de la carpeta “state” creamos un nuevo archivo para nuestro Reducer, lo llamaremos “product.reducer.ts”:



Definimos un Reducer exportando una constante. Dado que este Reducer escuchará eventos relacionados con las características de nuestro producto y transformará el estado del producto, lo llamaremos “productReducer”:

A continuación, creamos la función Reducer utilizando “createReducer” y añadimos el import necesario:

```
1  import { createReducer } from '@ngrx/store';
2
3  export const productReducer = createReducer();
```

El primer argumento que se pasa a la función “createReducer” es un objeto que especifica el estado inicial del Store. Dado que este Reducer se define para nuestro feature product, este argumento define los valores iniciales para el slice “products” del estado. En este punto, solo estamos implementando el flag “showProductCode”, así que vamos a establecer su valor inicial a true:

```
import { createReducer } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true }
);
```

A continuación, añadimos una función “on” y su import correspondiente. Definimos una función “on” para cada Action que maneja este Reducer. Hasta ahora, sólo tenemos un Action definido, “toggleProductCode”:

```
import { createReducer, on } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on()
);
```

El primer argumento de la función “on” es el creador del Action. Vamos a llamar “createAction” directamente aquí, añadiendo su import correspondiente:

```
import { createReducer, on, createAction } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on(createAction(''))
);
```

Para crear el Action, podemos pasar el nombre de nuestro Action en forma de un string. Este es el nombre que aparecerá en las herramientas para desarrolladores, así que especificaremos un nombre que sea útil al depurar:

```
import { createReducer, on, createAction } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on(createAction('[Product] Toggle Product Code'), )
);
```

El último argumento de la función “on” realiza el cambio de estado necesario para el Action especificado. Toma el estado actual del Store, realiza cualquier transformación necesaria y devuelve el nuevo estado al Store:

```
import { createReducer, on, createAction } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on(createAction('[Product] Toggle Product Code'), state => {
    return {
      showProductCode: !state.showProductCode
    };
  })
);
```

En nuestro ejemplo, activamos el flag “showProductCode”, invirtiendo su valor con el símbolo de exclamación “!”.

Pero hay un problema con este código. Si tenemos cualquier otro estado de producto en nuestro Store, lo reemplazaremos todo con esta única propiedad "showProductCode". Eso no es lo que queremos. En su lugar, primero extendemos el estado existente, haciendo un "spread" para hacer una copia, y luego aplicamos nuestros cambios a esa copia:

```
import { createReducer, on, createAction } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on(createAction('[Product] Toggle Product Code'), state => {
    return {
      ...state,
      showProductCode: !state.showProductCode
    };
  })
);
```

De forma temporal, vamos a añadir un poco de log para que podamos ver el estado. Vamos a mostrar el estado original antes de retornar el nuevo estado:

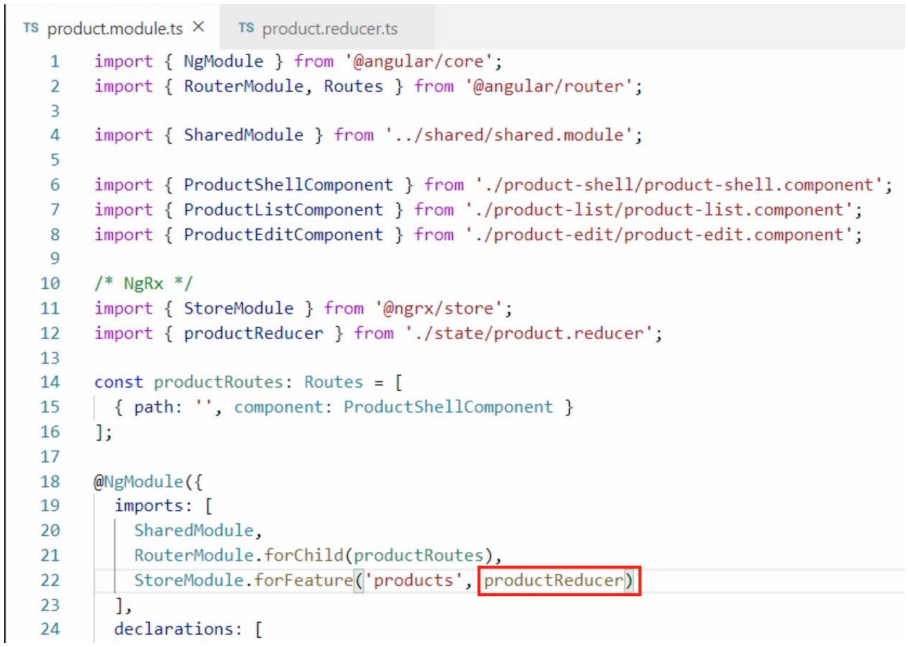
```
import { createReducer, on, createAction } from '@ngrx/store';

export const productReducer = createReducer(
  { showProductCode: true },
  on(createAction('[Product] Toggle Product Code'), state => {
    console.log('original state: ' + JSON.stringify(state));
    return {
      ...state,
      showProductCode: !state.showProductCode
    };
  })
);
```

Ahora que hemos construido el *productReducer*, vayamos a nuestro product.module y actualicemos el Store y actualicemos la inicialización del Store:

product.module.ts | product.reducer.ts

```
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3
4  import { SharedModule } from '../shared/shared.module';
5
6  import { ProductShellComponent } from './product-shell/product-shell.component';
7  import { ProductListComponent } from './product-list/product-list.component';
8  import { ProductEditComponent } from './product-edit/product-edit.component';
9
10 /* NgRx */
11 import { StoreModule } from '@ngrx/store';
12
13 const productRoutes: Routes = [
14   { path: '', component: ProductShellComponent }
15 ];
16
17 @NgModule({
18   imports: [
19     SharedModule,
20     RouterModule.forChild(productRoutes),
21     StoreModule.forFeature('products', {})
22   ]
```

Aquí, en lugar de un objeto vacío, haremos referencia a nuestro Reducer:



Ahora estamos listos para emitir nuestro Action.