

Título:

Guía Institucional de Redux Toolkit en React (Nivel Niño)

Luis

Autor: Luis Fernando Agudelo Gutiérrez

Fernando

Desarrollador de software | Bogotá, Colombia

Agudelo

Fecha: octubre 15/2025

Gutiérrez

Versión: v1.0 – Mi Documento técnico de apuntes

Índice técnico completo

NIVEL NIÑO - Fundamentos y estructura

Sección Título	Página
1.1 <i>npm install @reduxjs/toolkit react-redux -----</i>	6
2 <i>Estructura del proyecto y carpeta Store -----</i>	6
2.1 <i>Estructura básica del Slice -----</i>	7
2.2 <i>Importaciones-----</i>	8
2.3 <i>Estado inicial del Slice</i>	8
2.4 <i>Parte principal: reducers y exportaciones-----</i>	9-11
3 <i>Slice: definición, estructura y propósito -----</i>	12-13
4 <i>Utilizar Redux en el Main</i>	14
5 <i>Hook Custom consumiendo Redux de forma niño (novato)---</i>	15
5.1 <i>Funciones principales -----</i>	16
5.2 <i>Función para manejar Los cambios en el formulario de agregar-----</i>	17
5.3 <i>Funciones del Crud (dispatch)-(backend) -----</i>	18

NIVEL ADULTO - Formularios, validación y asincronía

Sección	Título	Página
	<i>Redux Nivel Adulto:</i>	
	<i>Manejo de formularios en React + Redux Toolkit: Desglose completo-----</i>	19-44
1	<i>Manejo de formularios con setProductoField-----</i>	19-24
1.1	<i>Cuando se debe utilizar-----</i>	19
1.2	<i>Ventajas-----</i>	19
1.3	<i>Desventajas-----</i>	19
1.4	<i>Ejemplo Real versión básica-----</i>	20-22
1.4.1	<i>Slice. -----</i>	20
1.4.1.1	<i>setProductoField (Función reducer dentro del slice) -----</i>	21
1.4.2	<i>Utilizarlo en hook Custom-----</i>	22
1.5	<i>Ejemplo Real versión optimizada-----</i>	23-24
1.5.1	<i>Utilizarlo en hook Custom-----</i>	23
1.5.2	<i>Mapearlo en el componente correspondiente-----</i>	24
2	<i>fieldConfigs para formularios dinámicos-----</i>	25-30
2.1	<i>Cuando se debe utilizar -----</i>	25
2.2	<i>Ventajas -----</i>	25
2.3	<i>Desventajas -----</i>	25
2.4	<i>Ejemplo Real-----</i>	26-30
2.4.1	<i>Estructura fieldConfigs.js-----</i>	26
2.4.2	<i>Configuración declarativa de campos (fieldConfigs) para formularios -----</i>	26
2.4.3	<i>Renderizado dinámico de formularios con fieldConfigs y Redux Toolkit -----</i>	27-30
3	<i>Formik + Yup + Redux Toolkit -----</i>	31-44
3.1	<i>Cuando se debe utilizar -----</i>	31
3.2	<i>Ventajas -----</i>	31
3.3	<i>Desventajas-----</i>	31
3.4	<i>Ejemplo real aplicado -----</i>	32-44
3.4.1	<i>Instalación Dependencias (Yup, Formik) -----</i>	32
3.4.2	<i>Crear el esquema de validación con Yup -----</i>	33
3.4.2.1	<i>Archivo YUP.js -----</i>	34-40
3.4.3	<i>Crear función en el Slice para actualizar varios campos a la vez -----</i>	40
3.4.4	<i>Implementar la Lógica en el hook Custom -----</i>	41-44

REDUX TOOLKIT AVANZADO - `createAsyncThunk` y `extraReducers`

Sección	Título	Página
4	<i>Redux Toolkit con <code>createAsyncThunk</code></i> -----	45-88
4.1	<i>¿Qué es?</i> -----	45
4.2	<i>¿Cuándo se utiliza?</i> -----	45
4.3	<i>Ventajas</i> -----	45
4.4	<i>Desventajas (y cómo mitigarlas)</i> -----	46
4.5	<i>Estructura básica de funcionamiento</i> -----	46-47
4.6	<i>Ejemplo real</i> -----	48
4.6.1	<i>Slice con La implementación correcta para <code>createAsyncThunk</code></i> -----	48
4.6.1.1	<i>Estados iniciales en el Slice</i> -----	49
4.6.1.2	<i>Thunks asíncronos (Listar)</i> -----	50-56
4.6.1.3	<i>Thunks asíncronos (Crear o Add)</i> -----	57
4.6.1.4	<i>Thunks asíncronos (update)</i> -----	58
4.6.1.5	<i>Thunks asíncronos (Delete)</i> -----	59
4.6.1.6	<i>Thunks asíncronos (Import multiple from excel)</i> -----	60
4.6.1.6	<i>Thunks asíncronos (Update multiple from excel)</i> -----	61
4.6.1.7	<i>SLICE Reducers + Acciones(Reducers Sincronos)</i> -----	62
4.6.1.8	<i>SLICE EXTRA REDUCERS (para thunks)</i> -----	63-75
4.6.1.8.1	<i>SLICE EXTRA REDUCERS (Crud Listar-estado-Pending)</i> -----	64-65
4.6.1.8.2	<i>SLICE EXTRA REDUCERS (Crud Listar-estado-fulfilled)</i> -----	66-67
4.6.1.8.3	<i>SLICE EXTRA REDUCERS (Crud Listar-estado-rejected)</i> -----	68
4.6.1.8.4	<i>SLICE EXTRA REDUCERS (Crud Create-Pending, fulfilled, rejected)</i> -----	69
4.6.1.8.5	<i>SLICE EXTRA REDUCERS (Crud Update-Pending, fulfilled, rejected)</i> -----	70

<i>Sección</i>	<i>Título</i>	<i>Página</i>
4.6.1.8.6	<i>SLICE EXTRA REDUCERS (Crud Delete-Pending,fulfilled,rejected)</i> -----	71
4.6.1.8.7	<i>SLICE EXTRA REDUCERS (Crud ImportProductos-Pending,fulfilled,rejected)</i> -----	72
4.6.1.8.8	<i>SLICE EXTRA REDUCERS (Crud UpdateMultiple-Pending,fulfilled,rejected)</i> -----	73
4.6.1.8.9	<i>SLICE Exportar Acciones y Reducers</i> -----	74
4.6.1.8.10	<i>Conclusión Slice</i> -----	75
4.6.1.9	<i>Usándolo en el hook Custom</i> -----	76-84
4.6.1.9.1	<i>Importar Las acciones y Reducer del slice</i> -----	76
4.6.1.9.2	<i>Funcion Dispatch y useSelector</i> -----	76
4.6.1.9.3	<i>Handler Genérico y useEffect cargar Los datos</i> --	77
4.6.1.9.4	<i>Crud Agregar en el hook Custom (Thunk)</i> -----	78
4.6.1.9.5	<i>Función onSubmit</i> -----	79
4.6.1.9.6	<i>handleEdit (prepara la edición.)</i> -----	80
4.6.1.9.7	<i>Crud Editar (handleUpdateProduct) Enviar al backend Los datos</i> -----	81-83
4.6.1.9.8	<i>Crud Eliminar (handleDelete) Eliminar info</i> -----	84

SELECTORES OPTIMIZADOS - Reselect

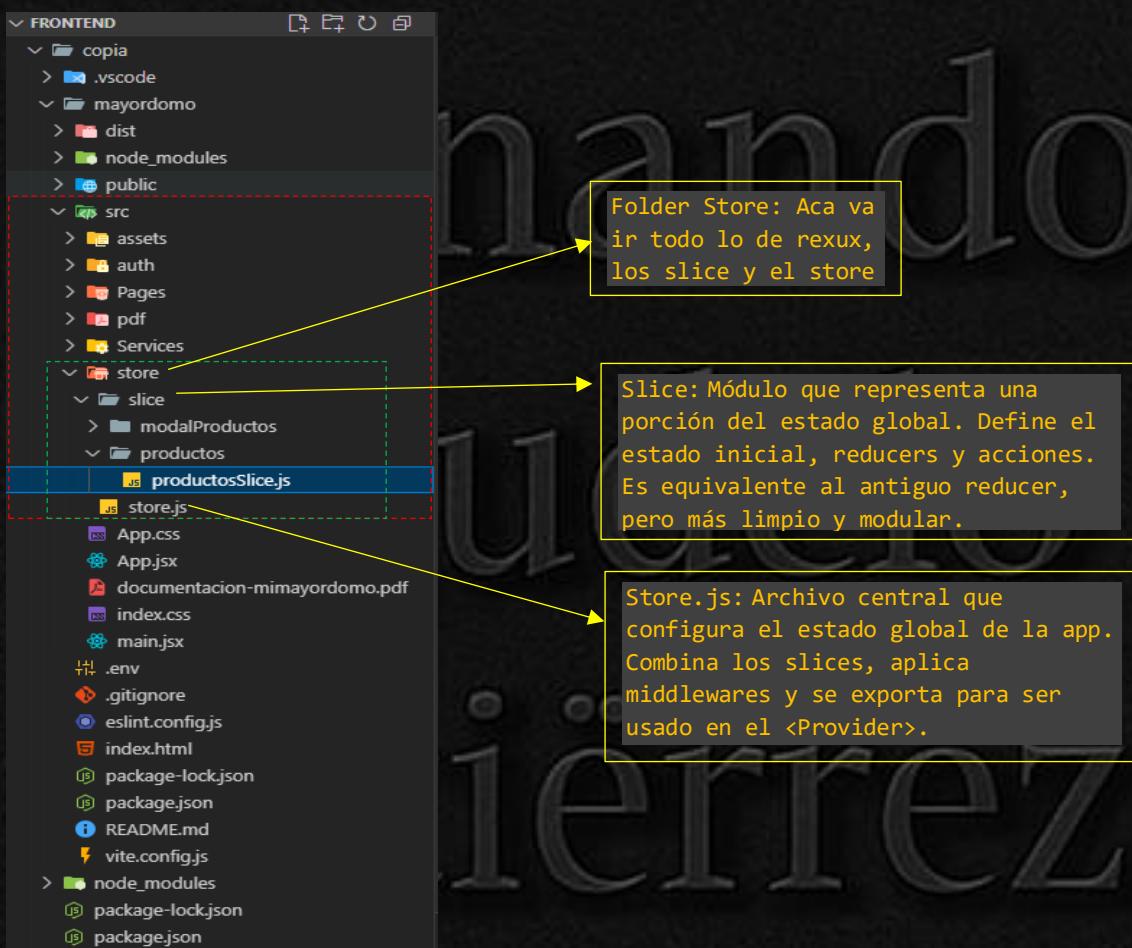
<i>Sección</i>	<i>Título</i>	<i>Página</i>
5	<i>¿Qué es Reselect?</i> -----	85
5.1	<i>¿Qué es Reselect?</i> -----	85
5.2	<i>Ventajas institucionales</i> -----	85
5.3	<i>¿Cuándo utilizarlo?</i> -----	85
5.4	<i>¿Cuándo no utilizarlo?</i> -----	85

REDUX (nivel niño)

1. Comando para instalar Redux Toolkit

```
npm install @reduxjs/toolkit react-redux
```

2. Estructura Empiezo con Redux nivel Niño



2.1 Estructura básica del Slice

Principal mente define el estado inicial, incluye Reducer y acciones. Por medio de este puedo tener un “estado global” por decirlo así. Como yo lo veo, seria tener un hook Custom y a la vez un hook Reducer en uno solo. Que en ese caso no se le pasa a un contex, si no al Store.

The screenshot shows a code editor with several tabs at the top: useCustomProductos.js, productosReducer.js, main.jsx, productosSlice.js (active), store.js, and TbodyProductos.jsx. The code in productosSlice.js is as follows:

```
copia > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createSlice } from "@reduxjs/toolkit"
2
3 // Estado inicial del formulario
4 const initialProducto = {
5 }
6
7
8 export const productosSlice = createSlice({
9   name: "productos",
10  initialState: initialProducto,
11  reducers: {
12    ...
13  },
14})
15
16 export const { ... } = productosSlice.actions
17
18 export default productosSlice.reducer
19
20
```

Annotations explain the code structure:

- Las importaciones**: Points to the first line: `import { createSlice } from "@reduxjs/toolkit"`.
- Los estados Iniciales del Slice**: Points to the `initialState` assignment in line 4.
- Parte principal, Define el nombre del slice, el estado inicial y los reducers que modifican el estado.**: Points to the `createSlice` call in line 8.
- Expone las acciones para ser usadas en componentes (dispatch) y el reducer para ser integrado en el store.**: Points to the `actions` and `reducer` exports in lines 16-18.
- El default export siempre debe ser el reducer.**: Points to the final `export default` statement in line 19.

2.2 Importaciones:

```
import { createSlice } from "@reduxjs/toolkit"
```

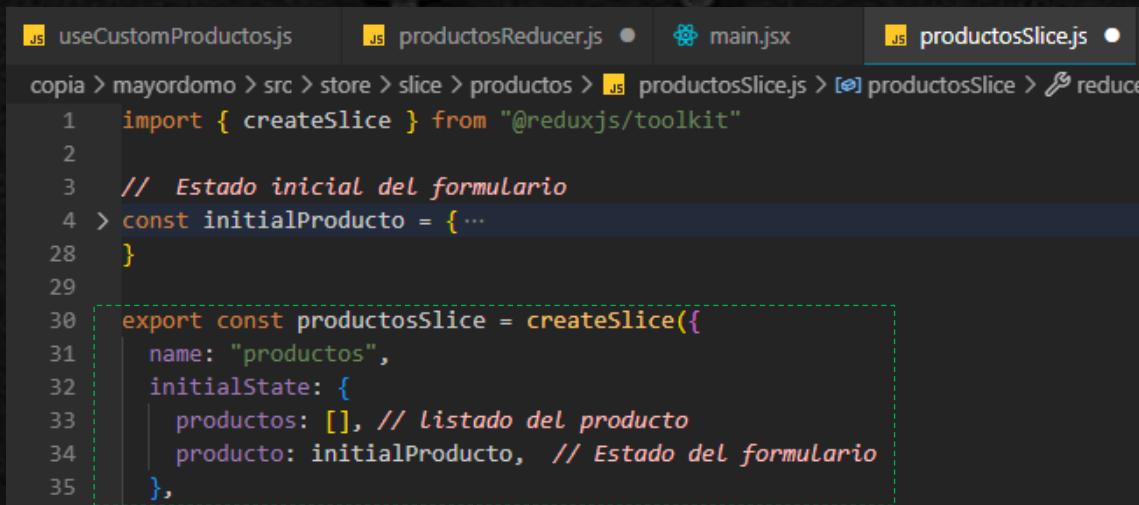
Esta parte es donde se importan todo lo requerido.

2.3 Los estados Iniciales del Slice

```
copia > mayordomo > src > store > slice > productos > productosSlice.js > ...
1  import { createSlice } from "@reduxjs/toolkit"
2
3  // Estado inicial del formulario
4  const initialProducto = {
5    id_productos: "",
6    id_subCategorias: "",
7    nombreSubCategoria: "",
8    nombre: "",
9    description: "",
10   img: "",
11   codigoProduct: "",
12   precio: 0.0,
13   descuento: 0.0,
14   marca: "",
15   contenido: 0,
16   presentacion: "",
17   unidadMedida: "",
18   destacado: false,
19   activo: false,
20   promociones: "",
21   iva: 0,
22   img1: "",
23   img2: "",
24   img3: "",
25   img4: "",
26   img5: "",
27   precioNuevo: 0,
28 }
29
30 > export const productosSlice = createSlice({ ...
31   })
32
33 > export const { ...
34   } = productosSlice.actions
35
36 > export default productosSlice.reducer
```

Este seria el estado
Inicial del formulario en
este caso para productos

2.4 Parte principal:



The screenshot shows a code editor with several tabs at the top: 'useCustomProductos.js', 'productosReducer.js', 'main.jsx', and 'productosSlice.js'. The 'productosSlice.js' tab is active. The code in the editor is as follows:

```
1 import { createSlice } from "@reduxjs/toolkit"
2
3 // Estado inicial del formulario
4 > const initialProducto = { ...
28 }
29
30 export const productosSlice = createSlice({
31   name: "productos",
32   initialState: {
33     productos: [], // Listado del producto
34     producto: initialProducto, // Estado del formulario
35   },
36 })
```

Parte principal: Recibe como parámetro un objeto.

Name:" Nombre slice"

initialState {

Objeto literal. Define la estructura base del estado que manejará este slice. Todo slice debe tener un initialState claro, reproducible y validado. Este objeto se convierte en la fuente de verdad para el estado local de este módulo.

}

Gutiérrez

```
useCustomProductos.js productosReducer.js main.js productosSlice.js store.js
copia > mayordomo > src > store > slice > productos > productosSlice.js > productosSlice > reducers > setProducto.js
1 import { createSlice } from "@reduxjs/toolkit"
2
3 // Estado inicial del formulario
4 const initialProducto = {
5   nombre: "Nuevo Producto",
6   precio: 100,
7   cantidad: 10
8 }
9
10 export const productosSlice = createSlice({
11   name: "productos",
12   initialState: {
13     productos: [], // Listado del producto
14     producto: initialProducto, // Estado del formulario
15   },
16
17   // ACA YA NO NECESITO MI HOOK REDUCECER AL MIGRAR A REDUX TOOLKIT,
18   // ESE REDUCER YA NO ES NECESARIO COMO FUNCION APARTE DE REDUX TOOLKIT
19   // YA ENCAPSULA TODO ESO DENTRO DEL CREATESLICE, QUE GENERA AUTOMATICAMENTE
20   // EL REDUCER Y LAS ACCIONES. SLICE YA GENERA EL REDUCER AUTOMATICAMENTE,
21   // Y LO INTEGRO EN EL STORE
22
23   reducers: {
24     // Cargar productos desde backend
25     loadProductos: (state, action) => {
26       state.productos.push(action.payload);
27     },
28     // Actualizar campo individual del formulario
29     setProductoField: (state, action) => {
30       const productoIndex = state.productos.findIndex(
31         (product) => product.id === action.payload.id
32       );
33
34       if (productoIndex !== -1) {
35         state.productos[productoIndex] = {
36           ...state.productos[productoIndex],
37           [action.payload.field]: action.payload.value
38         };
39       }
40     },
41     // Resetear el formulario
42     resetProducto: (state) => {
43       state.producto = initialProducto;
44     },
45     // Agregar producto
46     addProductos: (state, action) => {
47       state.productos.push(action.payload);
48     },
49     // Actualizar producto
50     updateProductos: (state, action) => {
51       const productoIndex = state.productos.findIndex(
52         (product) => product.id === action.payload.id
53       );
54
55       if (productoIndex !== -1) {
56         state.productos[productoIndex] = {
57           ...state.productos[productoIndex],
58           ...action.payload
59         };
60       }
61     },
62     // Eliminar producto
63     deleteProductos: (state, action) => {
64       state.productos = state.productos.filter(
65         (product) => product.id !== action.payload.id
66       );
67     },
68     // Actualizar múltiples productos
69     updateMultipleProductos: (state, action) => {
70       const productoIndex = state.productos.findIndex(
71         (product) => product.id === action.payload.id
72       );
73
74       if (productoIndex !== -1) {
75         state.productos[productoIndex] = {
76           ...state.productos[productoIndex],
77           ...action.payload
78         };
79       }
80     },
81   },
82 }
83
84 export const { addProducto, updateProducto, deleteProducto } = productosSlice.actions
85
86 export default productosSlice.reducer
```

Parte principal (reducer)

Reducers: {
 Acá va el Reducers en este caso el Crud,
}

```
copia > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createSlice } from "@reduxjs/toolkit"
2
3 // Estado inicial del formulario
4 > const initialProducto = { ...
28 }
29
30 > export const productosSlice = createSlice({ ...
87 })
88
89 export const {
90   loadProductos,
91   setProductoField,
92   resetProducto,
93   addProductos,
94   updateProductos,
95   deleteProductos,
96   updateMultipleProductos,
97 } = productosSlice.actions
98
99 export default productosSlice.reducer
100
```

Se exportan las acciones, casi igual que un hook Custom, solo que no por medio de un return,

El default export siempre debe ser el reducer.

Gutiérrez

3. Store (configuración básica)

```
copia > mayordomo > src > store > store.js > ...
1 import { configureStore } from '@reduxjs/toolkit'
2 import { //productosReducer,
3   productosSlice } from './slice/productos/productosSlice'
4
5
6
7 export const store = configureStore({
8   reducer: {
9
10    // forma de importar solo el reducer
11    // productos: productosReducer,
12
13    // forma de importar todo Slice
14    productos: productosSlice.reducer,
15
16  },
17}
18
```

Se importa la función principal para crear el store desde Redux Toolkit. Ya incluye configuración de middleware, DevTools y validaciones por defecto. No necesitas crear el store manualmente como en Redux clásico.

```
copia > mayordomo > src > store > store.js > ...
1 import { configureStore } from '@reduxjs/toolkit'
2 import { //productosReducer,
3   productosSlice } from './slice/productos/productosSlice'
4
5
6
7 export const store = configureStore({
8   reducer: {
9
10    // forma de importar solo el reducer
11    // productos: productosReducer,
12
13    // forma de importar todo Slice
14    productos: productosSlice.reducer,
15
16  },
17}
18
```

Se importa el slice completo desde su archivo. La línea comentada muestra una alternativa: importar solo el reducer si lo exportaras como productosReducer. En este caso, se importa el slice completo para acceder tanto al reducer como a las actions.

```
copia > mayordomo > src > store > store.js > ...
1 import { configureStore } from '@reduxjs/toolkit'
2 import { //productosReducer,
3   productosSlice } from './slice/productos/productosSlice'
4
5
6
7 export const store = configureStore({
8   reducer: {
9
10    // forma de importar solo el reducer
11    // productos: productosReducer,
12
13    // forma de importar todo Slice
14    productos: productosSlice.reducer,
15
16  },
17}
18
```

Se crea y exporta el store para que pueda ser usado en main.jsx dentro del <Provider>. Este store debe ser el único punto de entrada del estado global. Se recomienda documentar cada módulo que se integre.

useCustomProductos.js productosReducer.js main.jsx productosSlice.js store.js TbodyProductos.jsx

```
copia > mayordomo > src > store > store.js > ...
1 import { configureStore } from '@reduxjs/toolkit'
2 import { //productosReducer,
3   productosSlice } from './slice/productos/productosSlice'
4
5
6 export const store = configureStore({
7   reducer: {
8
9     // forma de importar solo el reducer
10    // productos: productosReducer,
11
12    // forma de importar todo Slice
13    productos: productosSlice.reducer,
14
15  },
16
17 })
18
```

Se define los reducers que componen el estado global.

Estructura: Cada propiedad representa una "slice" del estado. El nombre (productos) será la clave en el árbol de estado.

Si se exporta el reducer como `const productosReducer = productosSlice.reducer`, podrías usar esta forma.

Es útil si se quiere desacoplar el slice y el reducer en archivos separados.

Forma recomendada:

Se usa directamente el reducer del slice importado.

Esto permite mantener acceso a las actions desde el mismo archivo.

El nombre `productos` será la clave en el state, accesible como `state.productos`.

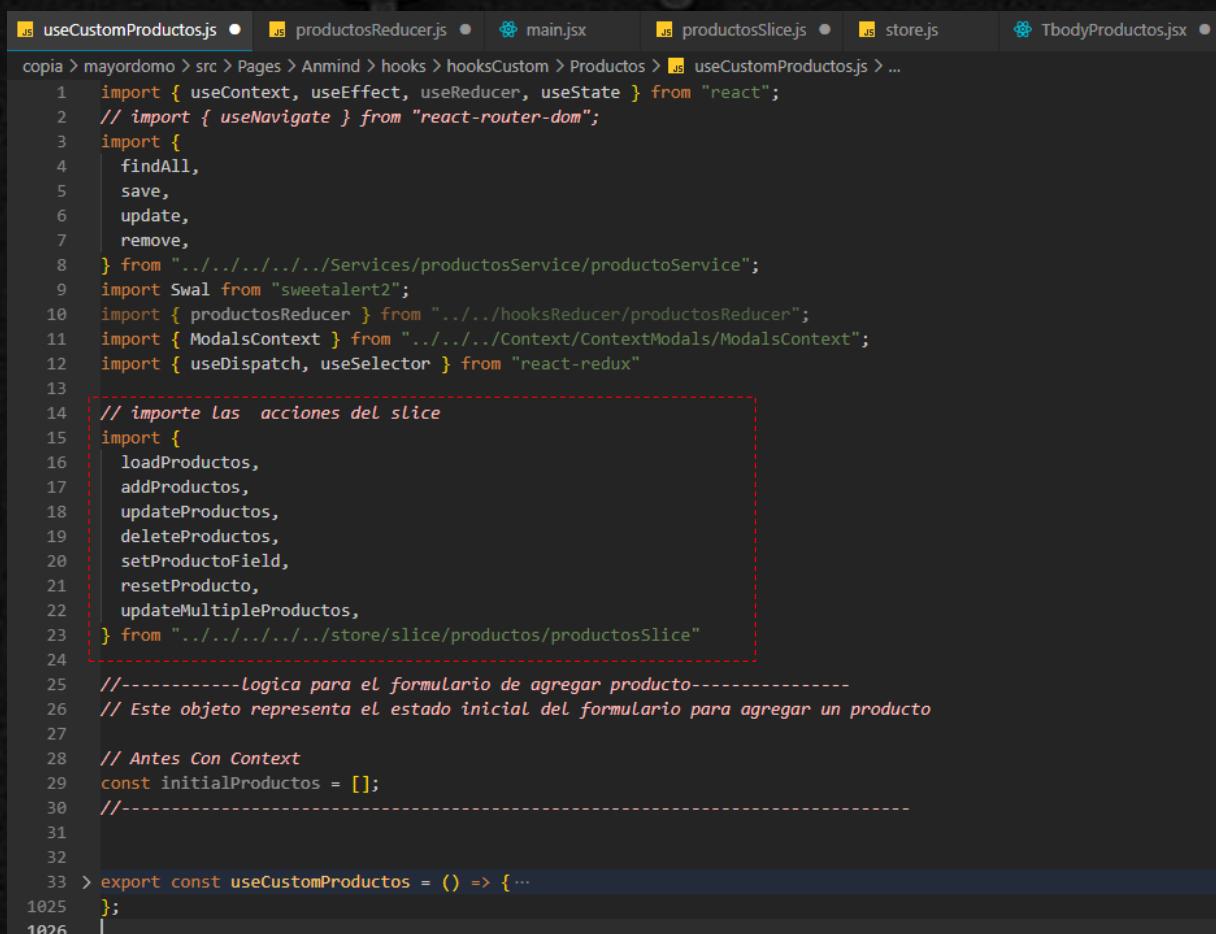
4. Utilizar Redux en el Main:

```
38 39  createRoot(document.getElementById("root")).render(  
40    <BrowserRouter>  
41      <StrictMode>  
42        <Provider store={  
43          store  
44        }>  
45          <DetallePedidosProvider>  
46            <FleteModalsProvider>  
47              <FletesProvider>  
48                <LoginProvider>  
49                  <BodyProvider>  
50                    <HeaderProvider>  
51                      <PedidosProvider>  
52                        <ComprasProvider>  
53                          <ModalsInventarioProvider>  
54                            <InventarioProvider>  
55                              <ModalsProvider>  
56                                <ProductosProvider>  
57                                  <SubCategoriasModalsProvider>  
58                                    <SubCategoriasProvider>  
59                                      <DescripcionProductoProvider>  
60                                        <CategoriasModalsProvider>  
61                                          <CategoriasProvider>  
62                                            <BuscarCategoriasProvider>  
63                                              <ModalsBannersProvider>  
64                                                <BannersProvider>  
65                                                  <PrincipalSistemaGestionProductsProvider>  
66                                                    <App />  
67                                                      </PrincipalSistemaGestionProductsProvider>  
68                                                    </BannersProvider>  
69                                              </ModalsBannersProvider>  
70                                            </BuscarCategoriasProvider>  
71                                          </CategoriasProvider>  
72                                            </CategoriasModalsProvider>  
73                                          </DescripcionProductoProvider>  
74                                            </SubCategoriasProvider>  
75                                          </SubCategoriasModalsProvider>  
76                                            </ProductosProvider>  
77                                              </ModalsProvider>  
78                                            </InventarioProvider>  
79                                              </ModalsInventarioProvider>  
80                                            </ComprasProvider>  
81                                              </PedidosProvider>  
82                                              </HeaderProvider>  
83                                              </BodyProvider>  
84                                              </LoginProvider>  
85                                              </FletesProvider>
```

Se utiliza el Provider de Redux y se le pasa el store. Con esto ya esta el contexto global, se simplifica más que como esta abajo, con context y me evito esa callback

5. Hook Custom consumiendo Redux de forma niño (novato):

Por mi parte soy fans de los hooks Custom, no hay como tener componentes libres de lógica o como dicen por ahí componentes limpios.



```
copia > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > ...
1  import { useContext, useEffect, useReducer, useState } from "react";
2 // import { useNavigate } from "react-router-dom";
3  import {
4    findAll,
5    save,
6    update,
7    remove,
8  } from "../../../../Services/productosService/productoService";
9  import Swal from "sweetalert2";
10 import { productosReducer } from "../../hooksReducer/productosReducer";
11 import { ModalsContext } from "../../Context/ContextModals/ModalsContext";
12 import { useDispatch, useSelector } from "react-redux"
13
14 // importe las acciones del slice
15 import {
16   loadProductos,
17   addProductos,
18   updateProductos,
19   deleteProductos,
20   setProductoField,
21   resetProducto,
22   updateMultipleProductos,
23 } from "../../../../store/slice/productos/productosSlice"
24
25 //-----Logica para el formulario de agregar producto-----
26 // Este objeto representa el estado inicial del formulario para agregar un producto
27
28 // Antes Con Context
29 const initialProductos = [];
30 //-
31
32
33 > export const useCustomProductos = () => { ...
1025 };
1026 |
```

Se importa las acciones del slice

5.1 Funciones principales. (`useSelector` permite leer el estado global del store desde cualquier componente o hook personalizado.)

```
export const useCustomProductos = () => [  
  
  // Importar el contexto para abrir el modal de edición  
  const { openProductModaledit } = useContext(ModalsContext);  
  
  // const navigate = useNavigate();  
  
  // Estado inicial para el formulario de agregar producto  
  //const [productos, dispatch] = useReducer(productosReducer, initialProductos);  
  
  // useDispatch() envía acciones al store  
  const dispatch = useDispatch()  
  
  //useSelector() Lee el estado del store y state.productos viene del nombre en configureStore  
  const { productos, producto } = useSelector(state => state.productos)
```

Hook de Redux Toolkit para enviar acciones al store.

Se usa para ejecutar cualquier acción definida en el slice (`productosSlice.actions`).

Evita tener que importar `store.dispatch` directamente.

Se recomienda encapsular cada acción en funciones claras dentro del hook (ej: `cargarProductos`, `resetearFormulario`, etc.).

Hook de Redux Toolkit para leer el estado global.

Accede a la porción `productos` del store, definida en `configureStore`.

`productos`: array con todos los productos.

`producto`: objeto individual del formulario.

```
//useSelector() Lee el estado del store y state.productos viene del nombre en configureStore  
const { productos, producto } = useSelector(state => state.productos)
```

copia > mayordomo > src > store > slice > productos > `productosSlice.js` > `productosSlice`

```
1  import { createSlice } from "@reduxjs/toolkit"  
2  
3  // Estado inicial del formulario  
4  > const initialProducto = {...  
28  }  
29  
30  export const productosSlice = createSlice({  
31    name: "productos",  
32    initialState: {  
33      productos: [], // Listado del producto  
34      producto: initialProducto, // Estado del formulario  
35    },  
36  })
```

5.2 Función para manejar los cambios en el formulario de agregar producto.

Cada handler actualiza el campo en Redux, despacha `setProductoField` con el campo y valor

(Dispatch permite enviar acciones al store para modificar el estado global)

```
//-----  
// Función para manejar los cambios en el formulario de agregar producto.  
  
// Cada handler actualiza el campo en Redux, despacha setProductoField con el campo y valor  
  
/* Antes con redux  
const handleNombre = (e) => {  
  setProducto((prev) => ({  
    ...prev,  
    nombre: e.target.value,  
  }));  
};*/  
const handleNombre = e => dispatch(setProductoField({ field: "nombre", value: e.target.value }))  
  
/* Antes con redux  
const handleDescription = (e) => {  
  setProducto((prev) => ({  
    ...prev,  
    description: e.target.value,  
  }));  
}; */  
const handleDescription = e => dispatch(setProductoField({ field: "description", value: e.target.value }))  
  
/* Antes con redux  
const handleCodigoProduct = (e) => {  
  setProducto((prev) => ({  
    ...prev,  
    codigoProduct: e.target.value,  
  }));  
};*/  
const handleCodigoProduct = e => dispatch(setProductoField({ field: "codigoProduct", value: e.target.value }))
```

```
export const productosSlice = createSlice({  
  name: "productos",  
  initialState: {  
    productos: [], // Listado del producto  
    producto: initialProducto, // Estado del formulario  
  },  
  
  // aca ya no necito mi hook reducer al migrar a Redux Toolkit,  
  // ese reducer ya no es necesario como función aparte Redux Toolkit  
  // ya encapsula todo eso dentro del createSlice, que genera automáticamente  
  // el reducer y las acciones. slice ya genera el reducer automáticamente  
  // y lo integra en el store  
  
  reducers: {  
    // Cargar productos desde backend  
    loadProductos: (state, action) => { ... },  
    // Actualizar campo individual del formulario  
    setProductoField: (state, action) => {  
      const { field, value } = action.payload  
      state.producto[field] = value  
    },  
    // Resetear el formulario  
    resetProducto: (state) => { ... },  
    // Agregar producto  
    addProductos: (state, action) => { ... },  
    // Actualizar producto  
    updateProductos: (state, action) => { ... },  
    // Eliminar producto  
    deleteProductos: (state, action) => { ... },  
    // Actualizar múltiples productos  
    updateMultipleProductos: (state, action) => { ... },  
  },  
});
```

Esta es una forma de niño (novata), solo que es muy repetitivo, tantas funciones handler, la idea de siempre es reutilizar, optimizar código. Siempre me pregunto de a ver una forma de que no haya necesidad de repetir todas esas funciones. por ahí investigando y de curioso Redux tiene para manejar formularios y se evita eso. Mas la adelante lo aprenderé nivel adulto

Acá el dispatch dispara la acción al store

ESTLI NIÑO (NOVATO)

5.3 Funciones del Crud (dispatch)-(backend) estas vienen del reducer del Slice creado

```
// Función para obtener todos los productos desde el backend

// Esta función se encarga de obtener todos los productos desde el backend
const getProductos = async () => {

  try {
    const result = await findAll();
    console.log("Datos Productos recibidos del backend:", result);

    // Asegurarse de que cada subcategoría tenga la propiedad categorias definida
    const ProductosNormalizados = result.map((producto) => ({
      ...producto,
    }));
  }

  // utilizando redux:
  dispatch(loadProductos(ProductosNormalizados));

  /* Antes con context:
  dispatch({
    type: "LoadProductos",
    payload: ProductosNormalizados,
 }); */

}

catch (error) {
  console.error("Error al cargar Productos:", error);
  Swal.fire("Error", "No se pudieron cargar los productos", "error");
}

};


```

El dispatch, manda la acción al store, así se utiliza con cada uno de los métodos del Crud.

Aclaro este tampoco me convenció, estoy mezclando lógica de negocio con el hook Custom,

La idea es que todo sea modularizado.

Pro hay investigando y de curioso si hay mas formas y mas profesionales nivel adulto. Mas adelante lo implementare:

Redux thunk y Redux RTK QQRy. Aprender esto para pasar a adulto

```
import { createSlice } from "@reduxjs/toolkit"

// Estado inicial del formulario
const initialProducto = { ... }

export const productosSlice = createSlice({
  name: "productos",
  initialState: { ... },
  reducers: {
    // Cargar productos desde backend
    loadProductos: (state, action) => { ... },
    // Actualizar campo individual del formulario
    setProductoField: (state, action) => { ... },
    // Resetear el formulario
    resetProducto: (state) => { ... },
    // Agregar producto
    addProductos: (state, action) => { ... },
    // Actualizar producto
    updateProductos: (state, action) => { ... },
    // Eliminar producto
    deleteProductos: (state, action) => { ... },
    // Actualizar múltiples productos
    updateMultipleProductos: (state, action) => { ... },
  },
})

export const { ... } = productosSlice.actions

export default productosSlice.reducer
```

•

Redux Nivel Adulto:

Manejo de formularios en React + Redux Toolkit: Desglose completo

1. setProductoField con dispatch

1.1 Cuando se debe utilizar:

Cuando el formulario necesita persistencia en el estado global.

Cuando se requiere trazabilidad entre vistas o componentes.

Cuando se quiere evitar useState local y migrar a Redux Toolkit.

Ideal para formularios que interactúan con backend (CRUD).

1.2 Ventajas:

Trazabilidad total del estado del formulario.

Integración directa con Redux Toolkit.

Permite centralizar la lógica en el slice.

Facilita la auditoría y el control de cambios.

Compatible con validación backend y sincronización externa.

1.3 Desventajas:

Requiere múltiples handlers si no se encapsula.

Puede volverse repetitivo en formularios grandes.

Mezcla lógica de UI con lógica de estado si no se usa hook Custom

1.4 Ejemplo Real versión básica:

1.4.1 Slice

```
copia > mayordomo > src > store > slice > productos > js productosSlice.js > ...
1  import { createSlice } from "@reduxjs/toolkit"
2
3  // Estado inicial del formulario
4  > const initialProducto = { ...
5  }
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30  export const productosSlice = createSlice({
31    name: "productos",
32    initialState: {
33      productos: [], // Listado del producto
34      producto: initialProducto, // Estado del formulario
35    },
36
37  // aca ya no necito mi hook reducecer al migrar a Redux Toolkit, ...
38
39
40
41
42
43    reducers: {
44      // Cargar productos desde backend
45      > loadProductos: (state, action) => { ...
46        },
47
48      // Actualizar campo individual del formulario
49      setProductoField: (state, action) => {
50        const { field, value } = action.payload
51        state.producto[field] = value
52      },
53      // Resetear el formulario
54      resetProducto: (state) => { ...
55        },
56
57      // Agregar producto
58      addProductos: (state, action) => { ...
59        },
60
61      // Actualizar producto
62      updateProductos: (state, action) => { ...
63        },
64
65      // Eliminar producto
66      deleteProductos: (state, action) => { ...
67        },
68
69      // Actualizar múltiples productos
70      updateMultipleProductos: (state, action) => { ...
71        },
72
73      },
74
75    }
76
77  )
78
79  > export const { ...
80    } = productosSlice.actions
81
82
83  export default productosSlice.reducer
```

1.4.1.1 setProductoField (Función reducer dentro del slice)

```
// Actualizar campo individual del formulario
setProductoField: (state, action) => {
  const { field, value } = action.payload
  state.producto[field] = value
},
```

Recibe dos parámetros:

- **state**: el estado actual del slice ({productos, producto}).
- **action**: el objeto que contiene type y payload.

Accede al objeto producto dentro del estado global.

Actualiza dinámicamente el campo indicado (field) con el nuevo valor (value).

Usa notación de corchetes para que el campo sea dinámico.

Desestructuración del objeto payload.

Field: nombre del campo que se quiere actualizar (ej. "nombre", "precio").

Value: nuevo valor que se asignará a ese campo.

1.4.2 Utilizarlo en hook Custom

```
export const useCustomProductos = () => {  
  // Importar el contexto para abrir el modal de edición  
  const { openProductModaledit } = useContext(ModalsContext);  
  
  // const navigate = useNavigate(); ...  
  const dispatch = useDispatch()  
  
  //useSelector() Lee el estado del store y state.productos viene del nombre en configureStore  
  const { productos, producto } = useSelector(state => state.productos)  
  
  /* Antes Con Context: ...  
  //-----  
  // Función para manejar los cambios en el formulario de agregar producto.  
  // Cada handler actualiza el campo en Redux, despacha setProductoField con el campo y valor  
  /* Antes con redux...  
  const handleNombre = e => dispatch(setProductoField({ field: "nombre", value: e.target.value }))  
  /* Antes con redux...  
  const handleDescription = e => dispatch(setProductoField({ field: "description", value: e.target.value }))  
  /* Antes con redux...  
  const handleCodigoProduct = e => dispatch(setProductoField({ field: "codigoProduct", value: e.target.value }))  
  
  /* Antes con redux...  
  const handlePrecio = e => dispatch(setProductoField({ field: "precio", value: parseFloat(e.target.value) }))  
  /* Antes con redux...  
  const handleIva = e => dispatch(setProductoField({ field: "iva", value: parseInt(e.target.value) || 0 }))  
  /* Antes con redux...  
  const handleDescuento = e => dispatch(setProductoField({ field: "descuento", value: parseFloat(e.target.value) }))
```

Esa es la forma de utilizarla, Claro está que, si es para 2 campos esta forma esta bien, pero si es mas de 2 utilizar la forma optimizada.

Y luego se mapea en el componente donde está el Form.

1.5 Ejemplo Real versión optimizada:

Se crean un handler genérico reutilizable que despacha setProductoField para cualquier campo del formulario, incluyendo checkbox.

Importante se debería crear un hook Custom para formularios y ahí se crearía la función genérica.

El slice es lo mismo, se utiliza la misma setProductoField

1.5.1 Utilizarlo en hook Custom

```
// Handler genérico optimizado
// Crea una función que recibe el nombre del campo a actualizar
const handleField = (field) =>
  // Devuelve una función que maneja el evento onChange del input
  (e) => {
    // Extrae el valor del input dependiendo del tipo:
    // Si es checkbox, usa e.target.checked (booleano)
    // Si es cualquier otro input, usa e.target.value (string, number, etc.)
    const value =
      e.target.type === "checkbox" ? e.target.checked : e.target.value;

    // Despacha La acción Redux para actualizar el campo en el estado global
    // El payload contiene el nombre del campo y su nuevo valor
    dispatch(setProductoField({ field, value }));
  };

// Despacha La acción Redux que reinicia el estado del formulario
// Ideal para limpiar el formulario después de enviar o cancelar
const resetFormulario = () => dispatch(resetProducto());

return [
  handleField,
  resetFormulario,
  productos, // Proporcionar el estado de productos
  producto, // Proporcionar el estado del producto
  updateProductosFromExcel, // actualizar con excel,
  /*handleNombre, // Proporcionar la función para manejar el estado de nombre...*/
  onSubmit, // Proporcionar la función de envío del formulario
  loadProductToEdit, // Proporcionar la función para cargar producto a editar
  updateProduct, // Proporcionar la función para actualizar el producto
  // Proporcionar la función para eliminar el producto
  handleDelete, // Proporcionar la función para manejar la eliminación del producto
  handleEdit, // Proporcionar la función para manejar la edición del producto
  deleteProduct, // Proporcionar la función para eliminar el producto
  // importProductosFromExcel,
  getImagenesProducto, // Proporcionar la función para obtener imágenes del producto,
  productoFiltrado,
  setProductoFiltrado,
  importProductosFromExcel,
];
};
```

1.5.2 Mapearlo en el componente correspondiente

```
6 import { useCustomProductos } from "../../../../../hooks/hooksCustom/Productos/useCustomProductos";
7
8 export const FormsUpdateProductos = () => {
9
10    const {
11        producto,
12        handleField,
13        updateProduct,
14        SubCategorias,
15    } = useCustomProductos();
16
17    const { subcategorias, getSubCategorias } = useContext(SubCategoriasContext)
18    useEffect(() => { ... }, []);
19
20    return [
21        <>
22            <div className="card">
23                <div className="card-header">
24                    <h5>Editar Producto</h5>
25                </div>
26                <div className="card-body">
27                    <form onSubmit={updateProduct}>
28
29                        /* Código y Nombre */
30                        <div className="row">
31                            <div className="col-md-6">
32                                <div className="mb-3">
33                                    <label htmlFor="codigo" className="form-label">...
34                                    </label>
35                                    <input
36                                        type="text"
37                                        className="form-control"
38                                        id="codigo"
39                                        name="codigo"
40                                        placeholder="Código producto"
41                                        required
42                                        value={producto.codigoProduct} // Asignar valor inicial
43                                        onChange={handleField("codigoProduct")}// Asegurarse de manejar el cambio
44                                    />
45                                </div>
46                            </div>
47                        </div>
48
49                    </form>
50                </div>
51            </div>
52        </>
53    ];
54}
```

Este se utiliza cuando son más de 2 campos y menos de 5

2. fieldConfigs para formularios dinámicos:

2.1 Cuando se debe utilizar:

Cuando el formulario tiene muchos campos y se quiere evitar JSX repetitivo.
Cuando se necesita validar, renderizar o modificar campos dinámicamente.

Cuando se busca una estructura declarativa y escalable.

Ideal para formularios que se generan desde backend o deben adaptarse por rol/perfil.

2.2 Ventajas:

Modularidad total: cada campo es un objeto con metadatos.

Reutilización: puedes usar la misma configuración en múltiples componentes.

Validación embebida: puedes definir required, min, max, pattern por campo.

Escalabilidad: ideal para formularios grandes o generados dinámicamente.

Limpieza visual: el JSX se reduce a un solo map () que renderiza todo.

2.3 Desventajas:

Requiere una función de renderizado que interprete los metadatos.

Puede ser más complejo de entender al principio.

Si no se documenta bien, puede perder trazabilidad en equipos grandes.

2.4 Ejemplo Real:

2.4.1 Estructura fielConfigs.js

```
✓ redux
  > .vscode
✓ mayordomo
  > dist
  > node_modules
  > public
✓ src
  > assets
  > auth
  > Pages
    > Anmind
      > configs
        > productoFieldConfigs.js
      > Context
      > hooks
      > pages
      > styles
      > Cliente
      > Services
      > store
      > App.css
      > App.jsx
      > index.css
      > main.jsx
    > .env
```

Se transforma cada campo del formulario en un objeto de configuración.

En lugar de escribir JSX manual para cada input, defines los campos como metadatos en un array (productoFieldConfigs) y se renderiza dinámicamente con map () .

2.4.2 Configuración declarativa de campos (fieldConfigs) para formularios

```
export const productoFieldConfigs = [
  { field: "codigoProduct", label: "Código", type: "text", required: true, placeholder: "Código del producto" },
  { field: "nombre", label: "Nombre", type: "text", required: true, placeholder: "Nombre del producto" },
  { field: "description", label: "Descripción", type: "textarea", required: true, placeholder: "Descripción del producto" },
  { field: "marca", label: "Marca", type: "text", required: true, placeholder: "Marca del producto" },
  { field: "contenido", label: "Contenido", type: "number", required: true, placeholder: "Contenido del producto" },
  { field: "unidadMedida", label: "Unidad de Medida", type: "text", required: true, placeholder: "Unidad de medida" },
  { field: "presentacion", label: "Presentación", type: "text", required: true, placeholder: "Presentación del producto" },
  { field: "precio", label: "Precio", type: "number", required: true, placeholder: "Precio del producto" },
  { field: "iva", label: "IVA", type: "number", required: true, placeholder: "IVA" },
  { field: "descuento", label: "Descuento", type: "number", required: false, placeholder: "Descuento aplicado" },
  { field: "promociones", label: "Promociones", type: "text", required: false, placeholder: "Promociones activas" },
  { field: "id_subCategorias", label: "Subcategoría", type: "select", required: true, options: [] }, // opciones dinámicas
  { field: "destacado", label: "Destacado", type: "checkbox" },
  { field: "activo", label: "Activo", type: "checkbox" },
  { field: "img", label: "Imagen principal", type: "text", required: true, placeholder: "URL imagen principal" },
  { field: "img1", label: "Imagen 1", type: "text", placeholder: "URL imagen 1" },
  { field: "img2", label: "Imagen 2", type: "text", placeholder: "URL imagen 2" },
  { field: "img3", label: "Imagen 3", type: "text", placeholder: "URL imagen 3" },
  { field: "img4", label: "Imagen 4", type: "text", placeholder: "URL imagen 4" },
  { field: "img5", label: "Imagen 5", type: "text", placeholder: "URL imagen 5" }
];
```

Propiedad	Función
field	Clave del estado en Redux Toolkit (<code>producto[field]</code>)
label	Texto visible en el formulario
type	Tipo de input (<code>text</code> , <code>number</code> , <code>textarea</code> , <code>checkbox</code> , <code>select</code>)
required	Si el campo es obligatorio (<code>true</code> o <code>false</code>)
placeholder	Texto guía dentro del input (solo para <code>text</code> , <code>number</code> , <code>textarea</code>)

Luis

2.4.3 Renderizado dinámico de formularios con fieldConfigs y Redux Toolkit

```
ix > mayordomo > src > Pages > Anmind > pages > PaginaSistemaGestionProducts > TablasSistemaGestionProducts > TablaProductos > FormsYModalsProd
import { useContext, useEffect } from "react";
import { SubCategoriasContext } from "../../../../../Context/ContextSubCategorias/SubCategoriasContext";
import { useCustomProductos } from "../../../../../hooks/hooksCustom/Productos/useCustomProductos";
import { productoFieldConfigs } from "../../../../../configs/productoFieldConfigs";

export const FormsUpdateProductos = () => {
  const { producto, handleField, updateProduct } = useCustomProductos();
  const { subcategorias, getSubCategorias } = useContext(SubCategoriasContext);

  useEffect(() => {
    // Inyectar opciones dinámicas al campo select
    const fieldConfigs = productoFieldConfigs.map(config =>
      config.field === "id_subCategorias"
        ? {
            ...config,
            // Se sobrescribe la propiedad "options" con el array de subcategorias obtenido del contexto
            // Esto permite que el <select> se renderice con las opciones reales disponibles en tiempo de ejecución
            options: subcategorias
          }
        : config
    );
  }, [ ]);

  return (
    ...
  );
};
```

```

return (
  <div className="card">
    <div className="card-header">...
    </div>
    <div className="card-body">
      <form onSubmit={updateProduct}>
        {fieldConfigs.map(config => {
          if (config.type === "select") {
            return (
              <div key={config.field} className="mb-3">
                <label htmlFor={config.field} className="form-label">{config.label}</label>
                <select
                  className="form-select"
                  id={config.field}
                  value={producto[config.field] || ""}
                  onChange={handleField(config.field)}
                  required={config.required}
                >
                  <option value="">Seleccione una opción</option>
                  {config.options.map(opt => (
                    <option key={opt.id_subCategorias} value={opt.id_subCategorias}>
                      {opt.nombre}
                    </option>
                  ))}
                </select>
              </div>
            );
          }
        });
      </form>
    </div>
  </div>
);

```

<div className="card-body">
 <form onSubmit={updateProduct}>

Se envía el formulario con updateProduct función que tengo en el hook Custom, que despacha la acción Redux para actualizar el producto.

{fieldConfigs.map(config => {

Itera sobre cada objeto de configuración en fieldConfigs, fieldConfigs es la función que se Creó atrás.

Cada config representa un campo del formulario con metadatos (field, label, type, etc.).

if (config.type === "select") {

Se verifica si el tipo de campo definido en fieldConfigs es "select".

Esto permite aplicar una lógica específica para renderizar un <select> en lugar de un <input>.

```

return (
  <div key={config.field} className="mb-3">
    <label htmlFor={config.field} className="form-label">{config.label}</label>

```

key={config.field} es obligatorio en. map () para identificar cada elemento de forma única en el DOM virtual dhtmlFor={config.field}

Se vincula la etiqueta con el <select> por su id, mejorando accesibilidad.

Renderiza la etiqueta del campo (label) usando el texto definido en config.label.

```

<select
  className="form-select"
  id={config.field}
  value={producto[config.field] || ""}
  onChange={handleField(config.field)}
  required={config.required}
>

```

id={config.field} lo vínculo con el label.

value={producto[config.field] || ""} extrae el valor actual desde Redux Toolkit (producto), o usa "" si está vacío.

onChange={handleField(config.field)} conecta el cambio con el handler genérico que despacha setProductoField. Esta viene de mi hook Custom y que a la vez del slice donde está el reducer setProductoField.

required={config.required} se aplica validación embebida si el campo es obligatorio.

Y si se necesita mas atributos se continua con la misma mecanica

```

return (
  <div className="card">
    <div className="card-header">...
    </div>
    <div className="card-body">
      <form onSubmit={updateProduct}>
        {fieldConfigs.map(config => {
          if (config.type === "select") {
            return (
              <div key={config.field} className="mb-3">
                <label htmlFor={config.field} className="form-label">{config.label}</label>
                <select
                  className="form-select"
                  id={config.field}
                  value={producto[config.field] || ""}
                  onChange={handleField(config.field)}
                  required={config.required}
                >
                  <option value="">Seleccione una opción</option>
                  {config.options.map(opt => (
                    <option key={opt.id_subCategorias} value={opt.id_subCategorias}>
                      {opt.nombre}
                    </option>
                  ))}
                </select>
              </div>
            );
          }
        });
      
```

```

<option value="">Seleccione una opción</option>
{config.options.map(opt => (
  <option key={opt.id_subCategorias} value={opt.id_subCategorias}>
    {opt.nombre}
  </option>

```

option value="" vacío para obligar al usuario a seleccionar una categoría válida, Mejora la {config.options.map(opt => ())} Itera sobre el array options definido en fieldConfigs. Este array fue inyectado dinámicamente desde el contexto de subcategorías.

```

<option key={opt.id_subCategorias} value={opt.id_subCategorias}>
  {opt.nombre}
</option>

```

})} Renderiza cada opción del <select> con su id_subCategorias como value y nombre como texto visible.

key={opt.id_subCategorias} asegura que React identifique cada opción de forma única.

```

<form onSubmit={updateProduct}>
  {fieldConfigs.map(config => {
    if (config.type === "select") {
      return (
        ...
      );
    }
    if (config.type === "checkbox") {
      return (
        ...
      );
    }
    if (config.type === "textarea") {
      return (
        ...
      );
    }
  });

```

Y con los demás es lo mismo, solo que en if se ledice si es igual que, ya sea select, checkbox, text area o number

```

>   <div className= "card" >
>     <div className="card-header">...
>     </div>
>     <div className="card-body" >
>       <form onSubmit={updateProduct}>
>         {fieldConfigs.map(config => {
>           if (config.type === "select") {
>             return (...);
>           }
>           if (config.type === "checkbox") {
>             return (...);
>           }
>           if (config.type === "textarea") {
>             return (...);
>           }
>         })
>
>         return (
>           <div key={config.field} className="mb-3">
>             <label htmlFor={config.field} className="form-label">{config.label}</label>
>             <input
>               type={config.type}
>               className="form-control"
>               id={config.field}
>               value={producto[config.field] || ""}
>               onChange={handleField(config.field)}
>               placeholder={config.placeholder}
>               required={config.required}
>             />
>           </div>
>         );
>       )}
>
>       /* Botones de acción */
>       <div className="d-flex justify-content-end mt-4"> ...
>       </div>
>     </form>
>   </div>
> </div>

```

`key={config.field}` es obligatorio en listas dinámicas para que React identifique cada elemento de forma única y evite errores de renderizado.

`<label htmlFor={config.field} className="form-label">{config.label}</label>`:

Renderiza la etiqueta (label) del campo, usando el texto definido en `config.label`.

`htmlFor={config.field}` vincula la etiqueta con el campo `<input>` por su id, mejorando accesibilidad y usabilidad.

`<input type={config.type}>`: Renderiza un campo `<input>` cuyo tipo (text, number, email, etc.) se define dinámicamente desde `config.type`.

Esto permite que el mismo bloque renderice múltiples tipos de campos sin duplicar lógica.

`id={config.field}`: Asigna un id único al campo, igual al nombre del campo (`field`).

Esto permite vincularlo correctamente con su label y facilita pruebas automatizadas.

`value={producto[config.field] || ""}`: Extrae el valor actual del campo desde el estado global `producto`. Si el valor no existe aún, se usa `""` como fallback para evitar errores de render.

`onChange={handleField(config.field)}`: Conecta el campo con el handler genérico `handleField`, que despacha `setProductoField` al slice de Redux Toolkit. Esto garantiza trazabilidad y sincronización inmediata con el estado global.

`placeholder={config.placeholder}`: Muestra el texto guía dentro del input, definido en `config.placeholder`. Mejora la experiencia del usuario al indicar qué debe ingresar.

`required={config.required}`: Aplica validación embebida si el campo está marcado como obligatorio (true). ◊ Esto se interpreta directamente desde la configuración

3. Formik + Yup + Redux Toolkit:

3.1 Cuándo se debe utilizar:

Cuando se busca desacoplar lógica de formulario del componente visual.

Cuando se necesita validación automática y sincronización con estado global.

Cuando el formulario tiene múltiples campos y se quiere evitar JSX repetitivo.

Ideal para formularios que deben ser auditables, escalables y adaptables por rol/perfil.

Cuando se quiere mantener trazabilidad institucional con Redux Toolkit y slices.

3.2 Ventajas:

Estado desacoplado: Formik maneja el estado local, Redux sincroniza el global.

Validación declarativa: Integración con Yup permite definir reglas por campo.

Reutilización total: fieldConfigs se convierte en fuente única de verdad.

JSX limpio: .map() interpreta y renderiza todos los campos sin duplicación.

Escalabilidad profesional: Ideal para formularios masivos, multi-step o generados desde backend.

3.3 Desventajas:

Requiere entender bien la arquitectura de Formik y Redux Toolkit.

La validación con Yup puede ser compleja si no se documenta bien.

Si no se sincroniza correctamente, puede haber desalineación entre Formik y Redux.

3.4 Ejemplo real aplicado:

3.4.1 Instalación Dependencias

Formik ⚡ Maneja formularios de forma más simple

- **Problema que resuelve:** Sin Formik, debes crear muchos useState y onChange manualmente
- **Ventaja:** Automatiza el manejo de valores, validaciones y envío
- Es como un "asistente personal" para tus formularios.

```
# Instalar Formik (manejo de formularios)
npm install formik
```

```
# Instalar Yup (validación de esquemas)
npm install yup
```

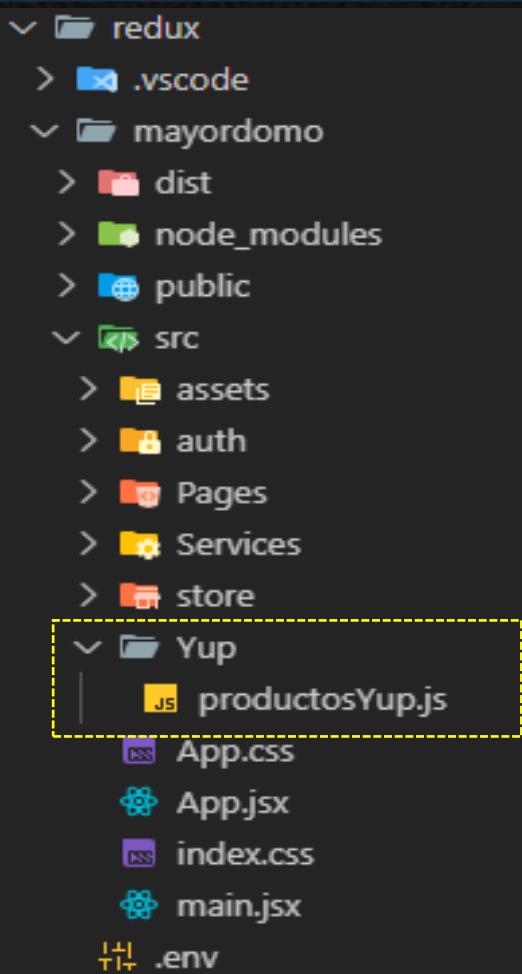
```
# Redux ya lo tienes instalado, pero asegúrate de tener:
npm install @reduxjs/toolkit react-redux
```

Yup Define reglas de validación

- **Problema que resuelve:** Validar datos manualmente es tedioso y propenso a errores
- **Ventaja:** Define reglas claras (ej: "el nombre es obligatorio", "el precio debe ser número")

Es como un "inspector de calidad" que revisa que todo esté correcto antes de enviar.

3.4.2 Crear el esquema de validación con Yup



3.4.2.1 Archivo YUP.js

```
import * as Yup from 'yup'; // se Importa toda la librería Yup

/**
 * ESQUEMA DE VALIDACIÓN CON YUP
 * Este esquema define las reglas que debe cumplir cada campo del formulario.
 * Es como una "lista de requisitos" que Yup verificará automáticamente.
 */

export const productoValidationSchema = Yup.object({}

    // Campo: Nombre del producto
    nombre: Yup.string() // Debe ser texto (string)
        .required('El nombre es obligatorio') // No puede estar vacío
        .min(3, 'El nombre debe tener al menos 3 caracteres') // Mínimo 3 letras
        .max(255, 'El nombre no puede exceder 255 caracteres'), // Máximo 255 letras

    // Campo: Descripción
    description: Yup.string()
        .required('La descripción es obligatoria')
        .min(10, 'La descripción debe tener al menos 10 caracteres')
        .max(255, 'La descripción no puede exceder 255 caracteres'),

    // Campo: Código del producto
    codigoProduct: Yup.string()
        .required('El código es obligatorio')
        .matches(/^[A-Z0-9-]+$/, 'El código solo puede contener letras mayúsculas, números y guiones'),

    // Campo: Precio
    precio: Yup.number() // Debe ser número
        .required('El precio es obligatorio')
        .positive('El precio debe ser mayor a 0') // No puede ser negativo
        .max(999999, 'El precio es demasiado alto'),

    // Campo: IVA
    iva: Yup.number()
        .required('El IVA es obligatorio')
        .oneOf([0, 5, 10, 19, 21], 'El IVA debe ser 0%, 5%, 10%, 19% o 21%'), // Solo estos valores
})
```

`import * as Yup from 'yup';`

Importa toda la librería Yup. Yup se usa para definir reglas de validación por campo, de forma declarativa.

`export const productoValidationSchema = Yup.object({`

Crea un objeto de validación con Yup. Cada propiedad representa un campo del formulario y sus reglas.

```

import * as Yup from 'yup'; // se Importa toda La librería Yup

/**
 * ESQUEMA DE VALIDACIÓN CON YUP
 * Este esquema define las reglas que debe cumplir cada campo del formulario.
 * Es como una "lista de requisitos" que Yup verificará automáticamente.
 */

export const productoValidationSchema = Yup.object([
    // Campo: Nombre del producto
    nombre: Yup.string() // Debe ser texto (string)
        .required('El nombre es obligatorio') // No puede estar vacío
        .min(3, 'El nombre debe tener al menos 3 caracteres') // Mínimo 3 letras
        .max(255, 'El nombre no puede exceder 255 caracteres'), // Máximo 255 letras

    // Campo: Descripción
    description: Yup.string()
        .required('La descripción es obligatoria')
        .min(10, 'La descripción debe tener al menos 10 caracteres')
        .max(255, 'La descripción no puede exceder 255 caracteres'),

    // Campo: Código del producto
    codigoProduct: Yup.string()
        .required('El código es obligatorio')
        .matches(/^[A-Z0-9-]+$/, 'El código solo puede contener letras mayúsculas, números y guiones'),

    // Campo: Precio
    precio: Yup.number() // Debe ser número
        .required('El precio es obligatorio')
        .positive('El precio debe ser mayor a 0') // No puede ser negativo
        .max(999999, 'El precio es demasiado alto'),

    // Campo: IVA
    iva: Yup.number()
        .required('El IVA es obligatorio')
        .oneOf([0, 5, 10, 19, 21], 'El IVA debe ser 0%, 5%, 10%, 19% o 21%'), // Solo estos valores
])

```

```

nombre: Yup.string()
    .required('El nombre es obligatorio')
    .min(3, 'El nombre debe tener al menos 3 caracteres')
    .max(255, 'El nombre no puede exceder 255 caracteres'),

```

nombre: Yup.string(): se debe especificar qué tipo de dato va manejar

.required('El nombre es obligatorio'): es un método de YUP. Que recibe un parámetro que queremos mostrar.

Indica que el campo no puede estar vacío ni undefined.

Mensaje: "El nombre es obligatorio" se mostrará si el campo está vacío.

Uso institucional: Garantiza que el campo sea obligatorio en el formulario, sin necesidad de validación manual.

Y así sucesivamente con los otros métodos que se requieran según el caso.

La misma dinámica para los otros campos del form

```
// Campos opcionales (imágenes)
img: Yup.string().url('Debe ser una URL válida').nullable(),
img1: Yup.string().url('Debe ser una URL válida').nullable(),
img2: Yup.string().url('Debe ser una URL válida').nullable(),
img3: Yup.string().url('Debe ser una URL válida').nullable(),
img4: Yup.string().url('Debe ser una URL válida').nullable(),
img5: Yup.string().url('Debe ser una URL válida').nullable(),
```

Campos de imagen deben ser URLs válidas.

Sonopcionales (nullable).

Fernando
Agudelo
Gutiérrez

Yup Data Types

Yup.string()

- .required (msg)**
Campo obligatorio
- .min(n, msg)**
Minimo de caracteres
- .max(n, msg)**
Maximo de caracteres
- .matches(regex, msg)**
Validacion por expresion regular
- .email(msg)**
Validacion de formato email
- .url(msg)**
Validacion de URL
- .lowercase()** Convierte a lowercase
- .uppercase()**, Convierte a uppercase
- .trim()** Elimina espacios

Yup.date()

- .required (msg)**
Minimo de caracteres
- .min(date, msg)**
Fecha minima

Yup.array()

- .required (msg)**
Campo obligatorio
- .min(n, msg)**
Minimo de elementos
- .max(n, msg)**
Maximo de elementos

Yup.boolean()

- .required (msg)**
Campo obligatorio
- .oneOf([true], msg)**
Debe ser verdadero
- .default(false)**
Valor por defecto

Yup.array()

- .required (msg)**
Campo obligatorio
- .test(name, msg, fn)**
Validacion personalizada
- .oneOf([...], msg)**
Debe coincidir con uno de
- .notOneOf([...], msg)**
No debe coincidir con

Yup Data Types

Yup.object()

- .shape { ... })
Define estructura interna
- .required (msg)
Mínimo de caracteres
- .noUnknown()
No permite claves no definidas

Yup.date()

- .required (msg)
Campo obligatorio
- .min(date, msg)
Fecha mínima
- .max(date, msg)
Fecha máxima

Yup.boolean()

- .required (msg)
Campo obligatorio
- .oneOf([true], msg)
Debe ser verdadero
- .default(false)
Valor por defecto

- .nullable()
Permite nula:
Valor por defecto

Yup.mixed()

- .required (msg)
Campo obligatorio
- .test(name, msg, fn)
Validación personalizada
- .oneOf(..., msg)
Debe coincidir con uno de
- .notOneOf(...,)
No debe coincidir con
- .default (value)

```
// Campo: Promociones
promociones: Yup.string().max(255, 'Las promociones no pueden exceder 255 caracteres').nullable(),

// Campo: Precio nuevo
precioNuevo: Yup.number()
  .min(0, 'El precio nuevo no puede ser negativo')
  .default(0),
);

/***
VALORES INICIALES DEL FORMULARIO
Estos son los valores por defecto cuando el formulario está vacío.
Formik usará esto como punto de partida.
*/
export const productoInitialValues = {
  id_productos: '',
  id_subCategorias: '',
  nombreSubCategoria: '',
  nombre: '',
  description: '',
  img: '',
  codigoProduct: '',
  precio: 0,
  descuento: 0,
  marca: '',
  contenido: 0,
  presentacion: '',
  unidadMedida: '',
  destacado: false,
  activo: false,
  promociones: '',
  iva: 0,
  img1: '',
  img2: '',
  img3: '',
  img4: '',
  img5: '',
  precioNuevo: 0,
};
```

Define los valores iniciales del formulario.

Formik usará esto como punto de partida.

Todos los campos están sincronizados con el esquema de Yup.

Esto asegura que el formulario tenga trazabilidad desde el inicio.

3.4.3 Crear función en el Slice para actualizar varios campos a la vez

```
export const productosSlice = createSlice({
  name: "productos",
  initialState: {
    productos: [], // Listado del producto
    producto: initialProducto, // Estado del formulario
  },
  // aca ya no necito mi hook reducecer al migrar a Redux dentro del createSlice, que genera automáticamente el reducer
  reducers: {
    // Cargar productos desde backend
    loadProductos: (state, action) => { ... },
    // Actualizar campo individual del formulario
    setProductoField: (state, action) => { ... },
    // Actualizar varios campos a la vez
    setProductoForEdit: (state, action) => {
      state.producto = { ...action.payload }
    },
    // Resetear el formulario
    resetProducto: (state) => { ... },
    // Agregar producto
  }
})
```

3.4.4 Implementar la lógica en el hook Custom

```
import { useContext, useEffect, useReducer, useState } from "react";
// import { useNavigate } from "react-router-dom";
import {
  findAll,
  save,
  update,
  remove,
} from "../../../../../Services/productosService/productoService";
import Swal from "sweetalert2";
import { productosReducer } from "../../hooksReducer/productosReducer";
import { ModalsContext } from "../../../../../Context/ContextModals/ModalsContext";
import { useDispatch, useSelector } from "react-redux"

// importe las acciones del slice
import {
  loadProductos,
  addProductos,
  updateProductos,
  deleteProductos,
  setProductoField,
  resetProducto,
  updateMultipleProductos,
  setProductoForEdit,
} from "../../../../store/slice/productos/productosSlice"

//-----Logica para el formulario de agregar producto-----
const initialProductos = [];
//-----
```



```
export const useCustomProductos = () => { ... }
```

Se importa la acción del reducer del Slice para actualizar varios campos a la vez

```
// ===== FUNCIONES DE EDICIÓN =====
const loadProductToEdit = (productoToEdit) => {
  dispatch(setProductoForEdit(productoToEdit))
}

const handleEdit = (producto) => {
  console.log("Cargando producto para editar:", producto)
  loadProductToEdit(producto)
  openProductModaledit()
}

// Función específica para manejar la actualización desde Formik ...
const handleUpdateProduct = async (values, formikHelpers) => { ... }
```

```

import { useCustomProductos } from "../../../../../hooks/hooksCustom/Productos/useCustomProductos"
import { Formik, Field, ErrorMessage, Form } from 'formik'
import { productoInitialValues, productoValidationSchema } from "../../../../../Yup/productoValidationSchema"

export const FormsUpdateProductos = () => {

  // Solo obtenemos datos y funciones del hook, sin lógica en el componente
  const { producto, handleUpdateProduct } = useCustomProductos()
  const { subcategorias, getSubCategorias } = useContext(SubCategoriasContext)

  useEffect(() => {
    ...
  }, [subcategorias, getSubCategorias])

  return (
    <div className="card shadow-sm">
      <div className="card-header bg-primary text-white">
        <h5 className="mb-0">
          <i className="bi bi-pencil-square me-2"></i>
          Editar Producto
        </h5>
      </div>

      <div className="card-body">
        /* NUEVO: Formik maneja todo el estado del formulario */
        /* enableReinitialize permite que el formulario se actualice cuando cambia 'producto' */
        <Formik
          enableReinitialize
          initialValues={producto || productoInitialValues}
          validationSchema={productoValidationSchema}
          onSubmit={handleUpdateProduct} // Función del hook que maneja la actualización
        >
          {({ isSubmitting, errors, touched, values }) => (
            <Form>...
              </Form>
            )
          }
        </Formik>
      </div>
    </div>
  )
}

```

Envolvemos el formulario en el componente `<Formik>` y recibe unos atributos:

`enableReinitialize`: sincroniza el formulario con Redux o backend sin tener que desmontar y volver a montar el componente.

`initialValues={producto}`: Define los **valores iniciales del formulario** y vienen del yup que cree. (es como el useState cuando ponía productos [" "]) es el estado inicial del formulario.

`onSubmit={handleUpdateProduct}`: Define la **función que se ejecuta al enviar el formulario** y viene del hook Custom

Desestructuración interna de pros. inyecta estas propiedades automáticamente:

Propiedad	Función
<code>isSubmitting</code>	Booleano que indica si el formulario está siendo enviado
<code>errors</code>	Objeto con los errores de validación generados por Yup
<code>touched</code>	Objeto que indica qué campos han sido tocados por el usuario
<code>values</code>	Objeto con los valores actuales del formulario

```

{({ isSubmitting, errors, touched, values }) => [
  <Form>
    /* Mostramos errores generales de submit si existen */
    {errors.submit && (
      <div className="alert alert-danger alert-dismissible fade show" role="alert">
        <i className="bi bi-exclamation-triangle-fill me-2"></i>
        <strong>Error:</strong> {errors.submit}
      </div>
    )}
  <div className="row">
    /* Código del Producto */
    <div className="col-md-6 mb-3">
      <label htmlFor="codigoProduct" className="form-label fw-semibold">
        Código del Producto <span className="text-danger">*</span>
      </label>
      /* Field de Formik maneja el valor y onChange automáticamente */
      <Field
        type="text"
        name="codigoProduct"
        id="codigoProduct"
        className={`form-control ${touched.codigoProduct && errors.codigoProduct ? 'is-invalid' : ''}`}
        placeholder="Ej: PROD-001"
      />
      /* ErrorMessage muestra el error de validación de Yup */
      <ErrorMessage name="codigoProduct" component="div" className="invalid-feedback" />
    </div>
    /* Nombre */
    <div className="col-md-6 mb-3"> ...
    </div>
  </div>
]

```

<Form> esta etiqueta no es de html, esta es de Formik asegurarse que al impórtala de de Formik

Condición que verifica si hay un error general de envío (submit).

Este tipo de error puede venir del backend o de lógica interna (ej. producto duplicado).

Gutiérrez

```
    Imágenes del Producto
    </h6>
  </div>
  /* Iteramos sobre las imágenes sin usar fieldConfigs */
  {[ 'img', 'img1', 'img2', 'img3', 'img4', 'img5' ].map((imgField, index) => (
    <div className="col-md-6 mb-3" key={imgField}>
      <label htmlFor={imgField} className="form-label fw-semibold">
        {index === 0 ? 'Imagen Principal' : `Imagen ${index}`}
      </label>
      <Field
        type="url"
        name={imgField}
        id={imgField}
        className="form-control"
        placeholder="https://ejemplo.com/imagen.jpg"
      />
      <ErrorMessage name={imgField} component="div" className="invalid-feedback" />
    </div>
  ))})
</div>
/* Botones */
<div className="d-flex justify-content-end gap-2 mt-4 pt-3 border-top">
  <button
    type="submit"
    className="btn btn-primary px-4"
    disabled={isSubmitting} // Formik maneja el estado de envío
  >
    /* Mostramos spinner mientras se envía */
    {isSubmitting ? (
      <>
        <span className="spinner-border spinner-border-sm me-2" role="status"></span>
        Actualizando...
      </>
    ) : (
      <>
        <i className="bi bi-check-circle me-2"></i>
        Actualizar Producto
      </>
    )}
  </button>
</div>
</Form>
```

Gutierrez

4. Redux Toolkit con `createAsyncThunk`:

4.1 ¿Qué es?:

Es una función oficial de Redux Toolkit que permite definir **acciones asincrónicas** (como llamadas a APIs) de forma declarativa y trazable. Automatiza el ciclo de vida de una petición:

`pending` → cuando inicia

`fulfilled` → cuando se resuelve exitosamente

`rejected` → cuando falla.

Se usa dentro de un slice para manejar lógica

asincrónica sin escribir reducers manuales para cada estado.

4.2 ¿Cuándo se utiliza?

Cuando necesitas **consultar datos desde un backend** (ej. productos, usuarios, etc.)

Para **crear, actualizar o eliminar** datos en una API

Cuando quieres **centralizar la lógica de carga, error y éxito** en Redux

En flujos donde el componente debe mostrar loading, error o datos según el estado

4.3 Ventajas institucionales:

Ventaja	Descripción
⌚ Ciclo de vida automático	Maneja <code>pending</code> , <code>fulfilled</code> , <code>rejected</code> sin escribir reducers manuales
📌 Código limpio	Evita lógica repetitiva en componentes
📌 Trazabilidad	Cada estado queda registrado en el store
✍️ Testable	Puedes simular cada fase en pruebas
📦 Encapsulable	Se puede integrar en hooks custom (<code>useProductos</code>)
⚠ Manejo de errores	Permite capturar errores y mostrarlos en UI
🚫 Integración con <code>extraReducers</code>	Actualiza el estado según el resultado de la acción

4.4 Desventajas (y cómo mitigarlas):

Desventaja	Mitigación institucional
💡 Verboseza en <code>extraReducers</code>	Usa <code>builder.addCase()</code> con funciones reutilizables
💡 Acoplamiento con Redux	Encapsula en hooks para evitar lógica en componentes
✖️ No reemplaza validaciones	Complementa con Yup o lógica en el backend
💡 Difícil de testear si no se estructura bien	Usa mocks y separa servicios en funciones puras

4.5 Estructura básica de funcionamiento



Bloque	Rol institucional
COMPONENTE REACT	Interfaz visual. Llama al hook y muestra datos.
HOOK CUSTOM <code>dispatch(fetchProductos)</code>	Encapsula lógica Redux. Expone funciones limpias
createAsyncThunk Servicio API	Dispara la acción asincrónica.
Slice	Ejecuta lógica, maneja estados 'pending', 'fulfilled', 'rejected'
STORE GLOBAL	Define cómo se actualiza el estado según el resultado
	Estado global accesible desde cualquier parte



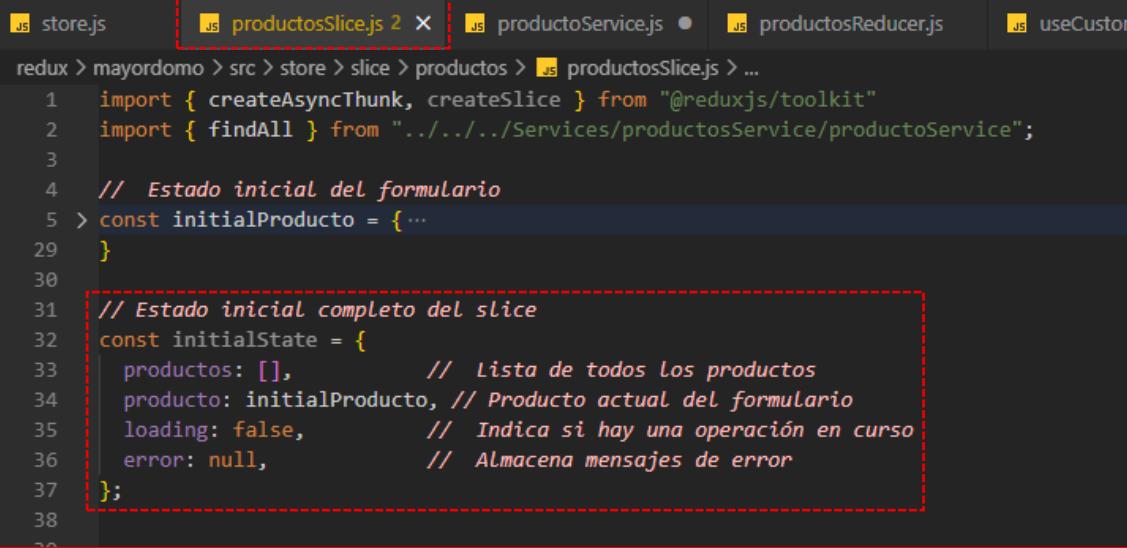
4.6 Ejemplo real:

4.6.1 Slice con la implementación correcta para createAsyncThunk:

```
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1  import { createSlice } from "@reduxjs/toolkit"
2
3  // Estado inicial del formulario
4  const initialProducto = {
5      id_productos: "",
6      id_subCategorias: "",
7      nombreSubCategoria: "",
8      nombre: "",
9      description: "",
10     img: "",
11     codigoProduct: "",
12     precio: 0.0,
13     descuento: 0.0,
14     marca: "",
15     contenido: 0,
16     presentacion: "",
17     unidadMedida: "",
18     destacado: false,
19     activo: false,
20     promociones: "",
21     iva: 0,
22     img1: "",
23     img2: "",
24     img3: "",
25     img4: "",
26     img5: "",
27     precioNuevo: 0,
28 }
29
30 > export const productosSlice = createSlice({
31     ...
32 })
33
34 > export const { ...
35     ...
36 } = productosSlice.actions
37
38 > export default productosSlice.reducer
39
40
```

según el caso, el estado inicial del formulario

4.6.1.1 Estados iniciales en el Slice



```
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productosService/productoService";
3
4 // Estado inicial del formulario
5 > const initialProducto = { ...
29 }
30
31 // Estado inicial completo del slice
32 const initialState = {
33   productos: [],           // Lista de todos los productos
34   producto: initialProducto, // Producto actual del formulario
35   loading: false,          // Indica si hay una operación en curso
36   error: null,             // Almacena mensajes de error
37 };
38
39
```

Implementamos, loading: que nos va indicar cuando una operación está en curso,

Error, importante para el manejo de errores

4.6.1.2 Thunks asíncronos (Listar)

```
store.js productosSlice.js 2 ● productoService.js ● productosReducer.js useCustomProductos.js
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productosService/productoService";
3
4 // Estado inicial del formulario
5 const initialProducto = ...
29 }
30
31 // Estado inicial completo del slice
32 const initialState = ...
37 };
38
39 //----- THUNKS ASÍNCRONOS -----
40
41 // Los thunks son funciones que ejecutan Lógica asíncrona.
42 // Automáticamente despiden acciones pending/fulfilled/rejected
43
44
45 // FETCH PRODUCTOS: Obtiene todos los productos desde el backend
46 /* Estados generados:
47 * - pending: Cuando inicia la petición
48 * - fulfilled: Cuando se obtienen los datos exitosamente
49 * - rejected: Cuando ocurre un error
50 */
51
52 export const fetchProductos = createAsyncThunk(
53   'productos/fetchProductos', // Nombre único de la acción
54   async (_, { rejectWithValue }) => {
55     // _ = no recibe parámetros
56     // rejectWithValue = función para manejar errores personalizados
57     try {
58       // Llama al service que hace la petición HTTP
59       const data = await findAll();
60       console.log("Productos obtenidos:", data);
61       return data; // Esto se envía a fulfilled
62     } catch (error) {
63       console.error(`Error al obtener productos: ${error}`);
64       // Extrae el mensaje de error y lo envía a rejected
65       return rejectWithValue(
66         error.response?.data?.message || error.message || "Error desconocido"
67       );
68     }
69   }
70 );
71 );
```

// Definimos una acción asíncrona con Redux Toolkit
export const fetchProductos = createAsyncThunk(

createAsyncThunk genera automáticamente tres acciones:

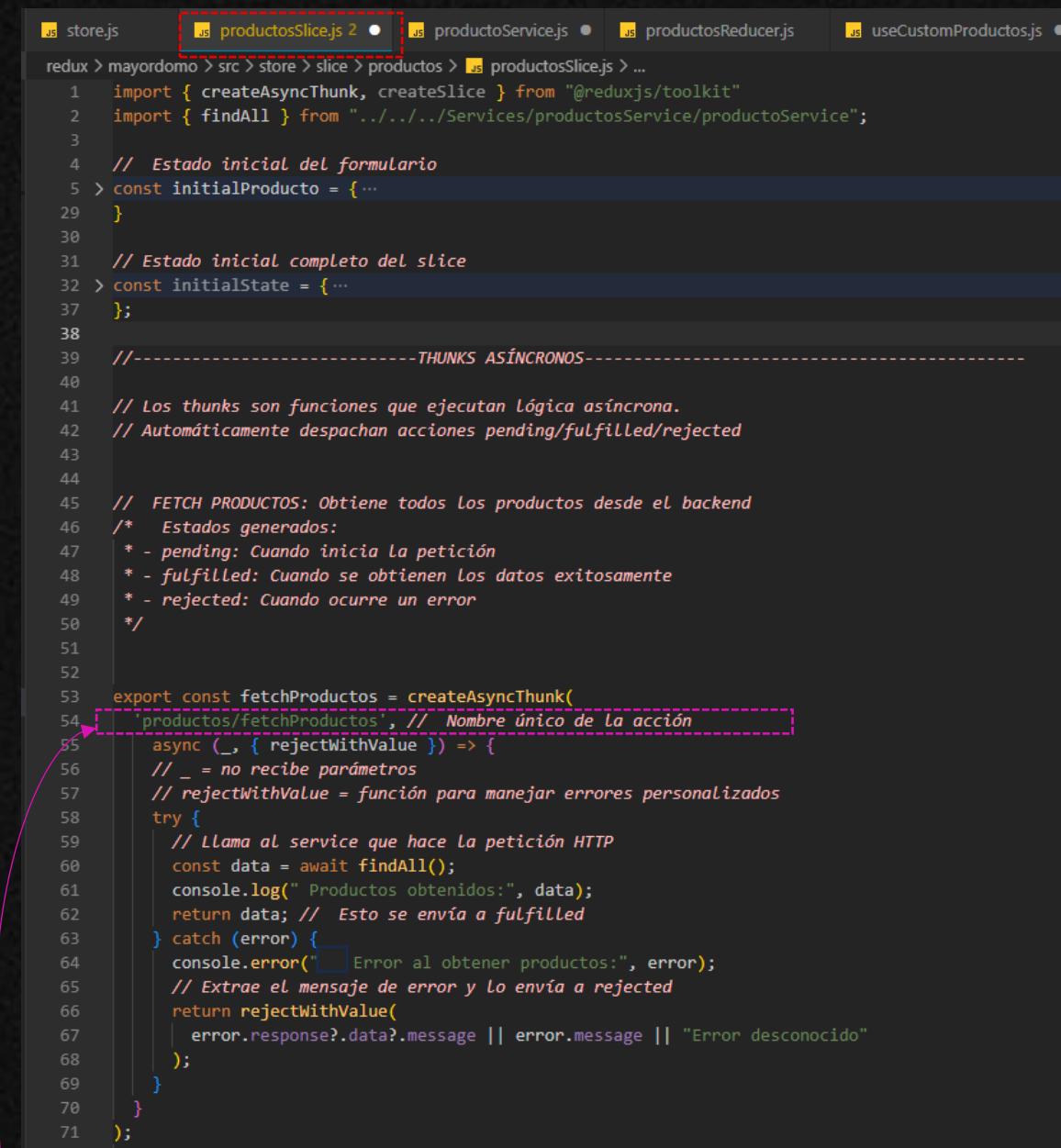
fetchProductos/pending

fetchProductos/fulfilled

fetchProductos/rejected

• Estas acciones serán manejadas en el slice mediante **extraReducers**.

En pocas palabras estamos creando el método Listar y Diciéndole que es igual a la función createAsyncThunk de redux toolkit. Y se divide por decirlo así en 3 pasos. Hay que importarla de redux/Toolkit



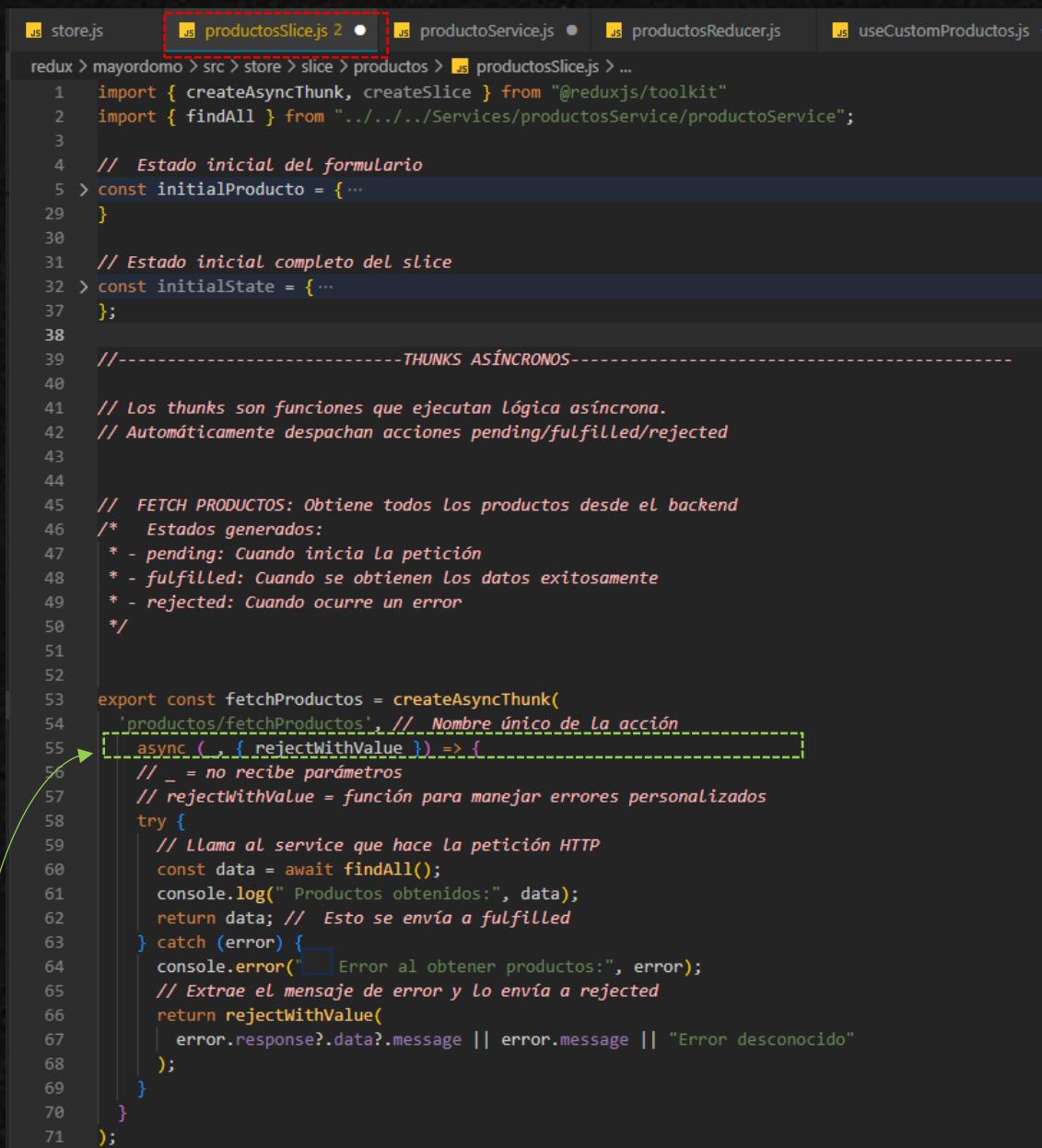
```
store.js productosSlice.js 2 ● productoService.js ● productosReducer.js ● useCustomProductos.js ●
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productoService/productoService";
3
4 // Estado inicial del formulario
5 const initialProducto = {...}
6
7 }
8
9 // Estado inicial completo del slice
10 const initialState = {...}
11 };
12
13 //----- THUNKS ASÍNCRONOS -----
14
15 // Los thunks son funciones que ejecutan lógica asíncrona.
16 // Automáticamente despachan acciones pending/fulfilled/rejected
17
18
19 // FETCH PRODUCTOS: Obtiene todos los productos desde el backend
20 /* Estados generados:
21 * - pending: Cuando inicia la petición
22 * - fulfilled: Cuando se obtienen los datos exitosamente
23 * - rejected: Cuando ocurre un error
24 */
25
26
27 export const fetchProductos = createAsyncThunk(
28   'productos/fetchProductos', // Nombre único de la acción
29   async (_, { rejectWithValue }) => {
30     // _ = no recibe parámetros
31     // rejectWithValue = función para manejar errores personalizados
32     try {
33       // Llama al service que hace la petición HTTP
34       const data = await findAll();
35       console.log("Productos obtenidos:", data);
36       return data; // Esto se envía a fulfilled
37     } catch (error) {
38       console.error("Error al obtener productos:", error);
39       // Extrae el mensaje de error y lo envía a rejected
40       return rejectWithValue(
41         error.response?.data?.message || error.message || "Error desconocido"
42       );
43     }
44   }
45 );
```

'**productos/fetchProductos**', // Nombre único de la acción

- Este string identifica la acción en el DevTools y en el slice.
- Sigue el patrón **"sliceName/actionName"** para trazabilidad.

simplemente el **nombre interno** de la acción que Redux Toolkit usa para:

- Generar los tipos de acción:
 - productos/fetchProductos/pending
 - productos/fetchProductos/fulfilled
 - productos/fetchProductos/rejected



```
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productosService/productoService";
3
4 // Estado inicial del formulario
5 const initialProducto = {...}
6
7
8 // Estado inicial completo del slice
9 const initialState = {...}
10
11
12 //-----THUNKS ASÍNCRONOS-----
13
14 // Los thunks son funciones que ejecutan lógica asíncrona.
15 // Automáticamente despachan acciones pending/fulfilled/rejected
16
17
18 // FETCH PRODUCTOS: Obtiene todos los productos desde el backend
19 /* Estados generados:
20 * - pending: Cuando inicia la petición
21 * - fulfilled: Cuando se obtienen los datos exitosamente
22 * - rejected: Cuando ocurre un error
23 */
24
25
26 export const fetchProductos = createAsyncThunk(
27   'productos/fetchProductos', // Nombre único de la acción
28   async (_, { rejectWithValue }) => {
29     // _ = no recibe parámetros
30     // rejectWithValue = función para manejar errores personalizados
31     try {
32       // Llama al servicio que hace la petición HTTP
33       const data = await findAll();
34       console.log(" Productos obtenidos:", data);
35       return data; // Esto se envía a fulfilled
36     } catch (error) {
37       console.error(" Error al obtener productos:", error);
38       // Extrae el mensaje de error y lo envía a rejected
39       return rejectWithValue(
40         error.response?.data?.message || error.message || "Error desconocido"
41       );
42     }
43   }
44 );
```

'productos/fetchProductos', // Nombre único de la acción

- Este string identifica la acción en el DevTools y en el slice.
- Sigue el patrón "sliceName/actionName" para trazabilidad.

Se le da el nombre de la acción

```

store.js  X productosSlice.js 2  ● productoService.js  ● productosReducer.js  ● useCustomProductos.js  ●
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productosService/productoService";
3
4 // Estado inicial del formulario
5 > const initialProducto = { ... }
29 }
30
31 // Estado inicial completo del slice
32 > const initialState = { ... }
37 };
38
39 > //-----THUNKS ASÍNCRONOS-----
46 /* Estados generados: ... */
51
52 export const fetchProductos = createAsyncThunk(
  'productos/fetchProductos', // Nombre único de la acción
54   async (_, { rejectWithValue }) => {
55     // _ = no recibe parámetros
56     // rejectWithValue = función para manejar errores personalizados
57     try {
58       // Llama al service que hace la petición HTTP
59       const data = await findAll();
60       console.log("Productos obtenidos:", data);
61       return data; // Esto se envía a fulfilled
62     } catch (error) {
63       console.error("Error al obtener productos:", error);
64       // Extrae el mensaje de error y lo envía a rejected
65       return rejectWithValue(
66         error.response?.data?.message || error.message || "Error desconocido"
67       );
68     }
69   }
70 );

```

`async (_, { rejectWithValue }) => {`

- ◆ Función asíncrona que ejecuta la lógica del thunk.
- ◆ `_` indica que no se reciben parámetros.
- ◆ `rejectWithValue` permite enviar errores personalizados al estado `rejected`.

Función asíncrona, como sabes siempre que hay `async`, debe de ir `await`.

Y recibe 2 parámetros:

Primer parámetro: Este parámetro representa los argumentos que recibe el thunk cuando se despacha.

En este caso `(_)`, significa que no recibe parámetros.

Porque `fetchProductos` está diseñado para consultar todos los productos, sin filtros ni condiciones.

No necesita saber id, ni nombre, ni categoría. Solo ejecuta `findAll()`.

Cuando el thunk **sí necesita parámetros**, este primer argumento puede ser:

Tipo	Ejemplo	Uso
<code>string</code>	<code>id</code>	Para obtener un producto por ID
<code>object</code>	<code>{ nombre, precio }</code>	Para crear o actualizar un producto
<code>array</code>	<code>[productos]</code>	Para importar productos en batch
<code>FormData</code>	<code>formData</code>	Para subir imágenes o archivos

Segundo parámetro: { rejectWithValue }:

Este es parte del **objeto** thunkAPI que Redux Toolkit inyecta automáticamente. Contiene utilidades como:

Propiedad	Función
<code>dispatch</code>	Permite despachar otras acciones desde dentro del thunk
<code>getState</code>	Permite acceder al estado global
<code>rejectWithValue</code>	Permite enviar errores personalizados al estado <code>rejected</code>
<code>extra</code>	Permite pasar configuraciones adicionales si se usa <code>extraArgument</code>

- En este caso, usamos solo `rejectWithValue` para manejar errores de forma controlada.

Conclusión institucional

El primer parámetro del thunk es **opcional** y depende del tipo de consulta.

Si no se usa, se marca como `_` por convención.

El segundo parámetro (`thunkAPI`) siempre está disponible y permite manejar errores, acceder al estado, y despachar otras acciones.

```
1 import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2 import { findAll } from "../../../../Services/productosService/productoService";
3
4 // Estado inicial del formulario
5 > const initialProducto = { ...
29 }
30
31 // Estado inicial completo del slice
32 > const initialState = { ...
37 };
38
39 > //-----THUNKS ASÍNCRONOS-----
46 > /* Estados generados: ...
51
52 export const fetchProductos = createAsyncThunk(
53   'productos/fetchProductos', // Nombre único de la acción
54   async (_, { rejectWithValue }) => {
55     // _ = no recibe parámetros
56     // rejectWithValue = función para manejar errores personalizados
57     try {
58       // Llama al service que hace la petición HTTP
59       const data = await findAll();
60       console.log(" Productos obtenidos:", data);
61       return data; // Esto se envía a fulfilled
62     } catch (error) {
63       console.error(` Error al obtener productos: ${error}`);
64       // Extrae el mensaje de error y lo envía a rejected
65       return rejectWithValue(
66         error.response?.data?.message || error.message || "Error desconocido"
67       );
68     }
69   }
70 );
```

El bloque try {} catch(error).

```
// Llama al service que hace la petición HTTP
const data = await findAll();
```

- `findAll()` es una función externa (service) que consulta la API.
- Se espera que retorne un array de productos.
- `await` pausa la ejecución hasta recibir respuesta.

Acá hace la petición Http al Service ya creado, para el método Listar.

```
console.log(" ✅ Productos obtenidos:", data);
```

Log institucional para trazabilidad en consola.

Permite verificar que los datos fueron recibidos correctamente.

```
return data; // ✅ Esto se envía a fulfilled
```

Si todo sale bien, se retorna `data`.

Redux despacha automáticamente `fetchProductos/fulfilled` con este payload.

```
store.js      x productosSlice.js 2 ● productoService.js ● productosReducer.js ● useCustomProductos.js ●
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
1  import { createAsyncThunk, createSlice } from "@reduxjs/toolkit"
2  import { findAll } from "../../../../Services/productosService/productoService";
3
4  // Estado inicial del formulario
5  const initialProducto = {...}
6
7  // Estado inicial completo del slice
8  const initialState = {...}
9
10 // -----
11 //----- THUNKS ASÍNCRONOS -----
12 /* Estados generados: ... */
13
14 export const fetchProductos = createAsyncThunk(
15   'productos/fetchProductos', // Nombre único de la acción
16   async (_, { rejectWithValue }) => {
17     // _ = no recibe parámetros
18     // rejectWithValue = función para manejar errores personalizados
19     try {
20       // Llama al service que hace la petición HTTP
21       const data = await findAll();
22       console.log(" Productos obtenidos:", data);
23       return data; // Esto se envía a fulfilled
24     } catch (error) {
25       console.error(` Error al obtener productos: ${error}`);
26       // Extrae el mensaje de error y lo envía a rejected
27       return rejectWithValue(
28         error.response?.data?.message || error.message || "Error desconocido"
29       );
30     }
31   }
32 );
```

```
// Extrae el mensaje de error y lo envía a rejected
return rejectWithValue(
  error.response?.data?.message || error.message || "Error desconocido"
);
```

- `rejectWithValue()` permite enviar un mensaje personalizado al estado `rejected`.
- Se prioriza el mensaje del backend (`error.response.data.message`), luego el genérico (`error.message`), y como último recurso, `"Error desconocido"`.

4.6.1.3 Thunks asíncronos (Crear o Add)

```
// Crud (Listar)
> export const fetchProductos = createAsyncThunk( ...  
);  
  
// Crud (Añadir) Crea un nuevo producto en el backend
export const createProducto = createAsyncThunk(  
  'productos/createProducto',  
  async (producto, { rejectWithValue }) => {  
    try {  
      // Transformar datos al formato que espera el backend  
      const productoToSave = {  
        ...producto,  
        // El backend espera un objeto subcategorias anidado  
        subcategorias: {  
          id_subCategorias: producto.id_subCategorias,  
          nombreSubCategoria: producto.nombreSubCategoria,  
        },  
      };  
  
      console.log(" Enviando producto:", productoToSave);  
      const data = await save(productoToSave);  
      console.log(" Producto creado:", data);  
      return data; // Producto con ID asignado por el backend  
    } catch (error) {  
      console.error(" Error al crear producto:", error);  
      return rejectWithValue(  
        error.response?.data?.message || error.message || "Error al crear producto"  
      );  
    }  
  }  
);
```

Es igual casi todos los otros métodos, solo que en unos sí recibe el primer parámetro, es importante crear copia del objeto, inmutabilidad ...

Gutiérrez

4.6.1.4 Thunks asíncronos (update)

```
export const createProducto = createAsyncThunk(...  
);  
  
// Crud (Actualizar) Actualiza un producto existente  
export const updateProductoThunk = createAsyncThunk(  
  'productos/updateProducto',  
  async (producto, { rejectWithValue }) => {  
    try {  
      // Asegurar que el ID esté presente  
      if (!producto.id_productos) {  
        throw new Error("El ID del producto es requerido para actualizar");  
      }  
  
      // Transformar datos al formato del backend  
      const productoToUpdate = {  
        ...producto,  
        subcategorias: {  
          id_subCategorias: producto.id_subCategorias,  
          nombreSubCategoria: producto.nombreSubCategoria,  
        },  
      };  
  
      console.log(" Actualizando producto:", productoToUpdate);  
      const data = await update(productoToUpdate);  
      console.log(" Producto actualizado:", data);  
      return data;  
    } catch (error) {  
      console.error(" Error al actualizar producto:", error);  
      return rejectWithValue(  
        error.response?.data?.message || error.message || "Error al actualizar producto"  
      );  
    }  
  }  
);
```

Gutiérrez

4.6.1.5 Thunks asíncronos (Delete)

```
store.js          productosSlice.js 2 X  productoService.js  ●  productosReducer.js  useCustomProduc
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
39 > //----- -THUNKS ASÍNCRONOS-----
40 > /* Estados generados: ...
41
42 // Crud (Listar)
43 > export const fetchProductos = createAsyncThunk( ...
44   );
45
46 // Crud (Create o Add) Crea un nuevo producto en el backend
47 > export const createProducto = createAsyncThunk( ...
48   );
49
50 // Crud (Update) Actualiza un producto existente
51 > export const updateProductoThunk = createAsyncThunk( ...
52   );
53
54 // Crud (Delete) Elimina un producto existente
55 > export const deleteProductoThunk = createAsyncThunk(
56   'productos/deleteProducto',
57   async (id, { rejectWithValue }) => {
58     try {
59       console.log(" Eliminando producto ID:", id);
60       await remove(id);
61       console.log(" Producto eliminado");
62       return id; // Devolvemos el ID para eliminarlo del estado
63     } catch (error) {
64       console.error(" Error al eliminar producto:", error);
65       return rejectWithValue(
66         error.response?.data?.message || error.message || "Error al eliminar producto"
67       );
68     }
69   }
70 );
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
```

ASADERO
Gutiérrez

4.6.1.6 Thunks asíncronos (Import multiple from excel)

The screenshot shows a code editor with several tabs at the top: 'store.js', 'productosSlice.js 2', 'productoService.js', 'productosReducer.js', and 'useCustomProductos.js'. The current file is 'productosSlice.js'. The code is part of a Redux slice for 'productos' and includes several asynchronous thunks:

```
53 > export const fetchProductos = createAsyncThunk( ...
71   );
72
73   // Crud (Create o Add) Crea un nuevo producto en el backend
74 > export const createProducto = createAsyncThunk( ...
99   );
100
101  // Crud (Update) Actualiza un producto existente
102 > export const updateProductoThunk = createAsyncThunk( ...
131   );
132
133  // Crud (Delete) Elimina un producto existente
134 > export const deleteProductoThunk = createAsyncThunk( ...
149   );
150
151 // Crud (Import multiple from excel) Importar multiples productos desde excel
152 export const importProductosThunk = createAsyncThunk(
153   'productos/importProductos',
154   async (productsData, { rejectWithValue }) => {
155     try {
156       console.log(` Importando ${productsData.length} productos...`);
157
158       // saveBatch procesa cada producto individualmente
159       const results = await saveBatch(productsData);
160
161       // Contar éxitos y errores
162       const successCount = results.filter(r => r.success).length;
163       const errorCount = results.filter(r => !r.success).length;
164
165       console.log(` Importados: ${successCount} | Errores: ${errorCount}`);
166
167       return {
168         results,
169         successCount,
170         errorCount
171       };
172     } catch (error) {
173       console.error(" Error en importación batch:", error);
174       return rejectWithValue(
175         error.response?.data?.message || error.message || "Error al importar productos"
176       );
177     }
178   }
179 );
180
```

A red dashed box highlights the code for the `importProductosThunk` function, which handles the import of multiple products from Excel.

4.6.1.6 Thunks asíncronos (Update multiple from excel)

```
JS store.js JS productosSlice.js 2 ● JS productoService.js ● JS productosReducer.js ● JS useCustomProductos.js ● JS produ
redux > mayordomo > src > store > slice > productos > JS productosSlice.js > ⚡ updateMultipleProductosThunk > ⚡ createAsyncThunk('productos/
182 // Crud (Update multiple from excel) Actualizar multiples productos desde excel
183 export const updateMultipleProductosThunk = createAsyncThunk(
184   'productos/updateMultipleProductos',
185   async (productsData, { getState, rejectWithValue }) => {
186     try {
187       // getState() permite acceder al estado actual de Redux
188       const { productos } = getState().productos;
189
190       console.log(` Actualizando ${productsData.length} productos...`);
191
192       let successCount = 0;
193       let errorCount = 0;
194       let notFoundCount = 0;
195       const updatedProducts = [];
196       const errors = [];
197       const notFound = [];
198
199       // Procesar cada producto
200       for (const productoData of productsData) {
201         try {
202           // Buscar producto existente por código
203           const existingProduct = productos.find(
204             p => String(p.codigoProduct).trim().toUpperCase() ===
205               String(productoData.codigoProduct).trim().toUpperCase()
206           );
207
208           if (!existingProduct) {
209             notFoundCount++;
210             notFound.push({
211               codigo: productoData.codigoProduct,
212               nombre: productoData.nombre
213             });
214             continue;
215           }
216
217           // Preparar producto para actualizar
218           const productoToUpdate = {
219             ...existingProduct,
220             ...productoData,
221             id_productos: existingProduct.id_productos, // Mantener ID original
222             subcategorias: {
223               id_subCategorias: productoData.id_subCategorias || existingProduct.id_subCategorias,
224               nombreSubCategoria: productoData.nombreSubCategoria || existingProduct.nombreSubCategoria,
225             },
226           };
227
228           // Actualizar en el backend
229           const response = await update(productoToUpdate);
230           updatedProducts.push(response);
231           successCount++;
232
233         } catch (error) {
234           errorCount++;
235           errors.push({
236             codigo: productoData.codigoProduct,
237             producto: productoData.nombre,
238             error: error.response?.data?.message || error.message,
239           });
240         }
241       }
242
243       console.log(` ✅ Actualizados: ${successCount} | ❌ Errores: ${errorCount} | ⚠️ No encontrados: ${notFoundCount}`);
244
245       return {
246         updatedProducts,
247         successCount,
248         errorCount,
249         notFoundCount,
250         errors,
251         notFound
252       };
253     } catch (error) {
254       console.error(` Error en actualización múltiple: ${error}`);
255       return rejectWithValue(
256         error.response?.data?.message || error.message || "Error al actualizar productos"
257       );
258     }
259   }
260 }
```

4.6.1.7 SLICE Reducers + Acciones(Reducers Síncronos)

```
store.js          productosSlice.js ● productoService.js ● productosReducer.js ● useCustomProductos.js ● productosSlice.js
redux > mayordomo > src > store > slice > productos > productosSlice.js > productosSlice
181
182 // Crud (Update multiple from excel) Actualizar multiples productos desde excel
183 > export const updateMultipleProductosThunk = createAsyncThunk( ...
184   );
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269   //----- REDUCERS SÍNCRONOS -----
270   // Estos se ejecutan inmediatamente sin operaciones asíncronas
271   reducers: {
272     // Actualiza un campo individual del formulario
273     setProductoField: (state, action) => {
274       const { field, value } = action.payload;
275       // Caso especial: IVA debe ser entero
276       state.producto[field] = field === "iva" ? parseInt(value, 10) : value;
277     },
278
279     // Carga un producto completo para edición
280     setProductoForEdit: (state, action) => {
281       state.producto = { ...action.payload };
282       state.error = null; // Limpiar errores previos
283     },
284
285     // Resetea el formulario a su estado inicial
286     resetProducto: (state) => {
287       state.producto = initialState;
288       state.error = null;
289     },
290
291     // Actualiza múltiples productos en el estado local (Usado después de actualización masiva exitosa)
292     updateMultipleProductosLocal: (state, action) => {
293       const updatedProducts = action.payload;
294       updatedProducts.forEach(updatedProduct => {
295         const index = state.productos.findIndex(
296           p => p.id_productos === updatedProduct.id_productos
297         );
298         if (index >= 0) {
299           state.productos[index] = updatedProduct;
300         }
301       });
302     },
303   },
304 }
```

Esto ya lo vimos, ya quedaría un poco más simplificado, solo utilizamos estos.

createSlice:

- **Centraliza** el estado, los reducers y las acciones en un solo bloque.
- **Genera automáticamente** el reducer y las acciones síncronas (setProductoField, resetProducto, etc.).
- **Evita verbosidad:** no necesitas escribir switch ni case manuales.
- **Facilita trazabilidad:** cada acción tiene nombre claro (productos/setProductoField).

Se usa para manejar lógica **local y síncrona**: formularios, edición, reseteo, carga en memoria.

4.6.1.8 SLICE EXTRA REDUCERS (para thunks)

```
store.js      productosSlice.js X  productoService.js  productosReducer.js  useCustomP...
redux > mayordomo > src > store > slice > productos > productosSlice.js > productosSlice > extraReducers
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269   //----- REDUCERS SÍNCRONOS -----
270   // Estos se ejecutan inmediatamente sin operaciones asíncronas
271   reducers: { ... },
272
273   //-----EXTRA REDUCERS (para thunks)-----
274   // Estos manejan las acciones generadas automáticamente por createAsyncThunk
275   extraReducers: (builder) => [
276
277     // ====== FETCH PRODUCTOS ======
278     builder
279       // Cuando inicia la petición
280       .addCase(fetchProductos.pending, (state) => {
281         console.log(" Cargando productos...");
282         state.loading = true;
283         state.error = null;
284       })
285       // Cuando se obtienen los datos exitosamente
286       .addCase(fetchProductos.fulfilled, (state, action) => {
287         console.log(" Productos cargados:", action.payload.length);
288         state.loading = false;
289         state.productos = action.payload; // Actualizar lista de productos
290       })
291       // Cuando ocurre un error
292       .addCase(fetchProductos.rejected, (state, action) => {
293         console.error(" Error al cargar productos:", action.payload);
294         state.loading = false;
295         state.error = action.payload; // Guardar mensaje de error
296     });
297   ],
298 }
```

Este bloque forma parte del slice y se encarga de **reaccionar a las acciones generadas automáticamente por** `createAsyncThunk`. En este caso, responde al ciclo de vida del thunk `fetchProductos` que creamos más atrás, de igual manera para los otros thunks del curd, que tiene tres estados:

- `pending`: cuando inicia la petición
- `fulfilled`: cuando se recibe respuesta exitosa
- `rejected`: cuando ocurre un error

`extraReducers`: es una **propiedad fija** que reconoce `createSlice` de Redux Toolkit. No se puede cambiar su nombre. Sirve para **reaccionar a acciones asíncronas** como `fetchProductos`, `createProducto`, etc (los thunk creados).

`(builder) => { ... }` es una **función flecha** que recibe un objeto llamado `builder`. Este objeto permite **encadenar métodos** como `.addCase()` para definir cómo el estado debe cambiar cuando se ejecuta un thunk.

4.6.1.8.1 SLICE EXTRA REDUCERS (Crud Listar-estado-Pending)

```
storejs          productosSlice.js X  productoService.js  productosReducer.js  useCustomP
redux > mayordomo > src > store > slice > productos > productosSlice.js > extraReducers
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269   //----- REDUCERS SÍNCRONOS -----
270   // Estos se ejecutan inmediatamente sin operaciones asíncronas
271   reducers: { ... },
272
273   //-----EXTRA REDUCERS (para thunks)-----
274   // Estos manejan las acciones generadas automáticamente por createAsyncThunk
275   extraReducers: (builder) => {
276
277     // ====== FETCH PRODUCTOS ======
278     builder
279       // Cuando inicia la petición
280       .addCase(fetchProductos.pending, (state) => {
281         console.log(" Cargando productos...");
282         state.loading = true;
283         state.error = null;
284       })
285       // Cuando se obtienen los datos exitosamente
286       .addCase(fetchProductos.fulfilled, (state, action) => {
287         console.log(" Productos cargados:", action.payload.length);
288         state.loading = false;
289         state.productos = action.payload; // Actualizar lista de productos
290       })
291       // Cuando ocurre un error
292       .addCase(fetchProductos.rejected, (state, action) => {
293         console.error(" Error al cargar productos:", action.payload);
294         state.loading = false;
295         state.error = action.payload; // Guardar mensaje de error
296       });
297   }
298 }
```

builder
.addCase(fetchProductos.pending, (state) => {

◆ **builder**

- Es el **objeto que Redux Toolkit inyecta dentro de extraReducers**.
- Permite **encadenar métodos como .addCase()**, **.addMatcher()**, etc.
- Su función es “construir” las reglas que definen cómo el estado debe reaccionar a acciones externas.
- Aunque puedes renombrarlo, se recomienda mantener **builder** por convención.

◆ **.addCase(...)**

- Es un **método del builder** que define cómo el estado debe cambiar cuando se dispara una acción específica.
- En este caso, se usa para manejar el estado **pending** del thunk **fetchProductos**.
- Se puede usar múltiples veces para cubrir **pending**, **fulfilled** y **rejected**.

◆ **fetchProductos.pending**

- Es el **tipo de acción automática** que Redux Toolkit genera cuando se despacha el thunk **fetchProductos** y la petición está en curso.
- Internamente equivale a **'productos/fetchProductos/pending'**.
- Esta acción se dispara **antes de que el backend responda**.

◆ **(state) => { ... }**

- Es la **función reductora** que define cómo debe cambiar el estado cuando se recibe esta acción.
- Recibe el **state** actual como parámetro.

```
store.js          productosSlice.js X          productoService.js ●          productosReducer.js          useCustomF
redux > mayordomo > src > store > slice > productos > productosSlice.js > [e] productosSlice > extraReducers
202
203 //----- SLICE (Reducers + Acciones)-----
204
205 //----- REDUCERS SÍNCRONOS -----
206 // Estos se ejecutan inmediatamente sin operaciones asíncronas
207 > reducers: { ...
208
209 //-----EXTRA REDUCERS (para thunks)-----
210 // Estos manejan las acciones generadas automáticamente por createAsyncThunk
211 extraReducers: (builder) => {
212
213     // ====== FETCH PRODUCTOS ======
214     builder
215         // Cuando inicia la petición
216         .addCase(fetchProductos.pending, (state) => {
217             console.log(" Cargando productos... ");
218             state.loading = true;
219             state.error = null;
220         })
221         // Cuando se obtienen los datos exitosamente
222         .addCase(fetchProductos.fulfilled, (state, action) => {
223             console.log(" Productos cargados:", action.payload.length);
224             state.loading = false;
225             state.productos = action.payload; // Actualizar lista de productos
226         })
227         // Cuando ocurre un error
228         .addCase(fetchProductos.rejected, (state, action) => {
229             console.error(" Error al cargar productos:", action.payload);
230             state.loading = false;
231             state.error = action.payload; // Guardar mensaje de error
232         });
233
234 }
```

`console.log(" Cargando productos... ");`

- Log institucional para trazabilidad.
- Permite verificar en consola que la carga ha comenzado.
- Útil para debugging y auditoría.

`state.loading = true;`

- Activa el flag `loading` en el estado global.
- Este flag puede ser usado en la UI para mostrar spinners, desactivar botones, etc.
- Indica que hay una operación en curso.

`state.error = null;`

- Limpia cualquier error previo que pudiera estar en el estado.
- Prepara el estado para una nueva operación sin residuos de errores anteriores.

Cumple con tres funciones clave:

Acción	Propósito
Activar <code>loading</code>	Mostrar que hay una operación en curso
Limpiar <code>error</code>	Evitar mostrar errores antiguos
Log de consola	Trazabilidad y diagnóstico técnico

4.6.1.8.2 SLICE EXTRA REDUCERS (Crud Listar-estado-fulfilled)

```
store.js          productosSlice.js X      productoService.js      productosReducer.js      useCustomF
redux > mayordomo > src > store > slice > productos > productosSlice.js > productosSlice > extraReducers
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269   //----- REDUCERS SÍNCRONOS -----
270   // Estos se ejecutan inmediatamente sin operaciones asíncronas
271   reducers: { ... },
272
273   //-----EXTRA REDUCERS (para thunks)-----
274   // Estos manejan las acciones generadas automáticamente por createAsyncThunk
275   extraReducers: (builder) => {
276
277     // ====== FETCH PRODUCTOS ======
278     builder
279       // Cuando inicia la petición
280       .addCase(fetchProductos.pending, (state) => {
281         console.log(" Cargando productos...");
282         state.loading = true;
283         state.error = null;
284       })
285       // Cuando se obtienen los datos exitosamente
286       .addCase(fetchProductos.fulfilled, (state, action) => {
287         console.log(" Productos cargados:", action.payload.length);
288         state.loading = false;
289         state.productos = action.payload; // Actualizar lista de productos
290       })
291       // Cuando ocurre un error
292       .addCase(fetchProductos.rejected, (state, action) => {
293         console.error(" Error al cargar productos:", action.payload);
294         state.loading = false;
295         state.error = action.payload; // Guardar mensaje de error
296       });
297   }
298 }
```

◆ `builder`

- Objeto que permite encadenar `.addCase()` para definir cómo el estado debe reaccionar a cada acción asíncrona.
- Inyectado automáticamente por Redux Toolkit dentro de `extraReducers`.

◆ `.addCase(...)`

- Método que define qué hacer cuando se dispara una acción específica.
- En este caso, se está manejando el estado `fulfilled` del thunk `fetchProductos`.

◆ `fetchProductos.fulfilled`

- Acción generada automáticamente por Redux Toolkit cuando el thunk `fetchProductos` se resuelve exitosamente.
- Internamente equivale a `'productos/fetchProductos/fulfilled'`.

◆ `(state, action) => {`

- Función reductora que define cómo debe cambiar el estado.
- Recibe dos parámetros:
 - `state`: el estado actual del slice.
 - `action`: contiene el `payload` con los datos retornados por el backend.

```
store.js          productosSlice.js X  productoService.js  productosReducer.js  useCustomP
redux > mayordomo > src > store > slice > productos > productosSlice.js > [e] productosSlice > extraReducers
262
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269 //----- REDUCERS SÍNCRONOS -----
270 // Estos se ejecutan inmediatamente sin operaciones asíncronas
271 > reducers: { ...
272   },
273
274 //-----EXTRA REDUCERS (para thunks)-----
275 // Estos manejan las acciones generadas automáticamente por createAsyncThunk
276 extraReducers: (builder) => {
277
278   // ====== FETCH PRODUCTOS ======
279   builder
280     // Cuando inicia la petición
281     .addCase(fetchProductos.pending, (state) => {
282       console.log(" Cargando productos...");
283       state.loading = true;
284       state.error = null;
285     })
286     // Cuando se obtienen los datos exitosamente
287     .addCase(fetchProductos.fulfilled, (state, action) => {
288       console.log(" Productos cargados:", action.payload.length);
289       state.loading = false;
290       state.productos = action.payload; // Actualizar lista de productos
291     })
292     // Cuando ocurre un error
293     .addCase(fetchProductos.rejected, (state, action) => {
294       console.error(" Error al cargar productos:", action.payload);
295       state.loading = false;
296       state.error = action.payload; // Guardar mensaje de error
297     });
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
```

console.log(" Productos cargados:", action.payload.length);

Log institucional que muestra cuántos productos fueron recibidos.

Útil para auditoría, validación y trazabilidad.

state.loading = false;

Desactiva el flag `loading`, indicando que la operación terminó.

Permite que la UI deje de mostrar el spinner o indicador de carga.

state.productos = action.payload; // Actualizar lista de productos

Actualiza el array `productos` en el estado global con los datos recibidos.

Esto permite que la interfaz se renderice con la nueva información.

4.6.1.8.3 SLICE EXTRA REDUCERS (Crud Listar-estado-rejected)

```

store.js          productosSlice.js X  productoService.js ●  productosReducer.js  useCustom...
redux > mayordomo > src > store > slice > productos > productosSlice.js > [x] productosSlice > [y] extraReducers
262
263 //----- SLICE (Reducers + Acciones)-----
264
265 export const productosSlice = createSlice({
266   name: "productos",
267   initialState,
268
269   //----- REDUCERS SÍNCRONOS -----
270   // Estos se ejecutan inmediatamente sin operaciones asíncronas
271   reducers: { ... },
272
273   //-----EXTRA REDUCERS (para thunks)-----
274   // Estos manejan las acciones generadas automáticamente por createAsyncThunk
275   extraReducers: (builder) => {
276
277     // ====== FETCH PRODUCTOS ======
278     builder
279       // Cuando inicia la petición
280       .addCase(fetchProductos.pending, (state) => {
281         console.log(" Cargando productos...");
282         state.loading = true;
283         state.error = null;
284       })
285       // Cuando se obtienen los datos exitosamente
286       .addCase(fetchProductos.fulfilled, (state, action) => {
287         console.log(" Productos cargados:", action.payload.length);
288         state.loading = false;
289         state.productos = action.payload; // Actualizar lista de productos
290       })
291       // Cuando ocurre un error
292       .addCase(fetchProductos.rejected, (state, action) => {
293         console.error(" Error al cargar productos:", action.payload);
294         state.loading = false;
295         state.error = action.payload; // Guardar mensaje de error
296       });
297   }
298 }

```

fetchProductos.rejected

Acción generada automáticamente cuando el thunk falla (por error de red, backend, etc.).

Internamente equivale a `'productos/fetchProductos/rejected'`.

`(state, action) => {`

Función reductora que maneja el error.

`action.payload` contiene el mensaje personalizado enviado desde `rejectWithValue`.

`console.error(" Error al cargar productos:", action.payload);`

Log de error para diagnóstico técnico.

Muestra el mensaje de error recibido desde el backend o generado localmente.

`state.loading = false;`

Desactiva el flag `loading`, aunque la operación haya fallado.

Permite que la UI deje de mostrar el spinner.

`state.error = action.payload; // Guardar mensaje de error`

Guarda el mensaje de error en el estado.

Esto permite mostrar alertas o mensajes en la interfaz.

Estado del thunk	Acción	Efecto en el estado
<code>pending</code>	Inicio	<code>loading = true, error = null</code>
<code>fulfilled</code>	Éxito	<code>loading = false, productos = payload</code>
<code>rejected</code>	Error	<code>loading = false, error = payload</code>

4.6.1.8.4 SLICE EXTRA REDUCERS (Crud Create-Pending.fulfilled.rejected)

```
//-----EXTRA REDUCERS (para thunks)-----  
// Estos manejan las acciones generadas automáticamente por createAsyncThunk  
extraReducers: (builder) => {  
  
    // ====== FETCH PRODUCTOS ======  
    builder  
        // Cuando inicia la petición  
        .addCase(fetchProductos.pending, (state) => { ...  
    })  
        // Cuando se obtienen los datos exitosamente  
        .addCase(fetchProductos.fulfilled, (state, action) => { ...  
    })  
        // Cuando ocurre un error  
        .addCase(fetchProductos.rejected, (state, action) => { ...  
    });  
  
    // ====== CREATE PRODUCTO ======  
    builder  
        .addCase(createProducto.pending, (state) => {  
            console.log(" Creando producto...");  
            state.loading = true;  
            state.error = null;  
        })  
        .addCase(createProducto.fulfilled, (state, action) => {  
            console.log(" Producto creado:", action.payload);  
            state.loading = false;  
            state.productos.push(action.payload); // Agregar nuevo producto a la lista  
            state.producto = initialProducto; // Resetear formulario  
        })  
        .addCase(createProducto.rejected, (state, action) => {  
            console.error(" Error al crear producto:", action.payload);  
            state.loading = false;  
            state.error = action.payload;  
        });
```

Gutiérrez

4.6.1.8.5 SLICE EXTRA REDUCERS (Crud Update-Pending,fulfilled,rejected)

```
JS store.js JS productosSlice.js X JS productoService.js ● JS productosReducer.js JS U
redux > mayordomo > src > store > slice > productos > JS productosSlice.js > [o] productosSlice > ⚡ extraReducers
265   export const productosSlice = createSlice({
267     extraReducers: (builder) => {
268       ...
269     });
270
271     // ===== CREATE PRODUCTO =====
272     builder
273       .addCase(createProducto.pending, (state) => { ...
274         })
275       .addCase(createProducto.fulfilled, (state, action) => { ...
276         })
277       .addCase(createProducto.rejected, (state, action) => { ...
278         });
279
280
281     // ===== UPDATE PRODUCTO =====
282     builder
283       .addCase(updateProductoThunk.pending, (state) => {
284         console.log(" Actualizando producto...");
285         state.loading = true;
286         state.error = null;
287       })
288       .addCase(updateProductoThunk.fulfilled, (state, action) => {
289         console.log(" Producto actualizado:", action.payload);
290         state.loading = false;
291
292         // Buscar y reemplazar el producto actualizado en la lista
293         const index = state.productos.findIndex(
294           p => p.id_productos === action.payload.id_productos
295         );
296         if (index >= 0) {
297           state.productos[index] = action.payload;
298         }
299       })
300       .addCase(updateProductoThunk.rejected, (state, action) => {
301         console.error(" Error al actualizar producto:", action.payload);
302         state.loading = false;
303         state.error = action.payload;
304       });
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
```

4.6.1.8.6 SLICE EXTRA REDUCERS (Crud Delete-Pending,fulfilled,rejected)

```
JS store.js          JS productosSlice.js ●   JS productoService.js ●   JS productosReducer.js   JS useCustomPr
redux > mayordomo > src > store > slice > productos > JS productosSlice.js > ⚡ productosSlice > ⚡ extraReducers
265   export const productosSlice = createSlice({
307     extraReducers: (builder) => {
348
349       // ===== UPDATE PRODUCTO =====
350       builder
351       .addCase(updateProductoThunk.pending, (state) => { ... })
355       .addCase(updateProductoThunk.fulfilled, (state, action) => { ... })
367       .addCase(updateProductoThunk.rejected, (state, action) => { ... })
372     );
373
374     // ===== DELETE PRODUCTO =====
375     builder
376       .addCase(deleteProductoThunk.pending, (state) => {
377         console.log(" Eliminando producto...");
378         state.loading = true;
379         state.error = null;
380       })
381       .addCase(deleteProductoThunk.fulfilled, (state, action) => {
382         console.log(" Producto eliminado ID:", action.payload);
383         state.loading = false;
384
385         // Filtrar el producto eliminado de la lista
386         state.productos = state.productos.filter(
387           p => p.id_productos !== action.payload
388         );
389       })
390       .addCase(deleteProductoThunk.rejected, (state, action) => {
391         console.error(" Error al eliminar producto:", action.payload);
392         state.loading = false;
393         state.error = action.payload;
394       });
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
167
```

4.6.1.8.7 SLICE EXTRA REDUCERS (Crud ImportProductos-Pending,fulfilled,rejected)

```
store.js          productosSlice.js ●  productoService.js  ●  productosReducer.js  ●  useCustomPro
redux > mayordomo > src > store > slice > productos > productosSlice.js > [!] extraReducers
265   export const productosSlice = createSlice({
266     extraReducers: (builder) => {
267       ...
268
269       // ===== DELETE PRODUCTO =====
270       builder
271         .addCase(deleteProductoThunk.pending, (state) => { ... })
272         .addCase(deleteProductoThunk.fulfilled, (state, action) => { ... })
273         .addCase(deleteProductoThunk.rejected, (state, action) => { ... });
274
275
276       // ===== IMPORT PRODUCTOS =====
277       builder
278         .addCase(importProductosThunk.pending, (state) => {
279           console.log(" Importando productos...");
280           state.loading = true;
281           state.error = null;
282         })
283         .addCase(importProductosThunk.fulfilled, (state, action) => {
284           console.log(" Importación completada:", action.payload);
285           state.loading = false;
286           // No actualizamos la lista aquí, se hace con fetchProductos después
287         })
288         .addCase(importProductosThunk.rejected, (state, action) => {
289           console.error(" Error en importación:", action.payload);
290           state.loading = false;
291           state.error = action.payload;
292         });
293
294
295       // ===== UPDATE MULTIPLE PRODUCTOS =====
296       builder
297         .addCase(updateMultipleProductosThunk.pending, (state) => { ... })
298         .addCase(updateMultipleProductosThunk.fulfilled, (state, action) => { ... })
299         .addCase(updateMultipleProductosThunk.rejected, (state, action) => { ... });
300
301     },
302   );
303
```

4.6.1.8.8 SLICE EXTRA REDUCERS (Crud UpdateMultiple-Pending,fulfilled,rejected)

```
store.js          productosSlice.js ●  productoService.js ●  productosReducer.js  useCustomProduct
redux > mayordomo > src > store > slice > productos > productosSlice.js > [!] productosSlice > extraReducers
265   export const productosSlice = createSlice({
307     extraReducers: (builder) => {
308       ,
381     .addCase(deleteProductoThunk.fulfilled, (state, action) => { ...
389   })
390   .addCase(deleteProductoThunk.rejected, (state, action) => { ...
394   });
395
396   // ===== IMPORT PRODUCTOS =====
397   builder
398   .addCase(importProductosThunk.pending, (state) => { ...
402   })
403   .addCase(importProductosThunk.fulfilled, (state, action) => { ...
407   })
408   .addCase(importProductosThunk.rejected, (state, action) => { ...
412   });
413
414   // ===== UPDATE MULTIPLE PRODUCTOS =====
415   builder
416   .addCase(updateMultipleProductosThunk.pending, (state) => {
417     console.log(" Actualizando múltiples productos...");
418     state.loading = true;
419     state.error = null;
420   })
421   .addCase(updateMultipleProductosThunk.fulfilled, (state, action) => {
422     console.log(" Actualización múltiple completada:", action.payload);
423     state.loading = false;
424
425     // Actualizar productos en el estado Local
426     if (action.payload.updatedProducts.length > 0) {
427       action.payload.updatedProducts.forEach(updatedProduct => {
428         const index = state.productos.findIndex(
429           p => p.id_productos === updatedProduct.id_productos
430         );
431         if (index >= 0) {
432           state.productos[index] = updatedProduct;
433         }
434       });
435     }
436   })
437   .addCase(updateMultipleProductosThunk.rejected, (state, action) => {
438     console.error(" Error en actualización múltiple:", action.payload);
439     state.loading = false;
440     state.error = action.payload;
441   });
442   ],
443 });


```

4.6.1.8.9 SLICE Exportar Acciones y Reducers

```
store.js          productosSlice.js ●  productoService.js ●  productosReducer.js
redux > mayordomo > src > store > slice > productos > productosSlice.js > ...
265   export const productosSlice = createSlice({
307     extraReducers: (builder) => {
396       // ===== IMPORT PRODUCTOS =====
397       builder
398       .addCase(importProductosThunk.pending, (state) => { ... })
402       .addCase(importProductosThunk.fulfilled, (state, action) => { ... })
403       .addCase(importProductosThunk.rejected, (state, action) => { ... })
407       .addCase(updateMultipleProductosThunk.pending, (state) => { ... })
408       .addCase(updateMultipleProductosThunk.fulfilled, (state, action) => { ... })
412       .addCase(updateMultipleProductosThunk.rejected, (state, action) => { ... })
413     }
414     // ===== UPDATE MULTIPLE PRODUCTOS =====
415     builder
416     .addCase(updateMultipleProductosThunk.pending, (state) => { ... })
420     .addCase(updateMultipleProductosThunk.fulfilled, (state, action) => { ... })
421     .addCase(updateMultipleProductosThunk.rejected, (state, action) => { ... })
436     .addCase(updateMultipleProductosThunk.pending, (state) => { ... })
437     .addCase(updateMultipleProductosThunk.fulfilled, (state, action) => { ... })
441     .addCase(updateMultipleProductosThunk.rejected, (state, action) => { ... })
442   },
443 );
444
445
446 //----- EXPORTAR ACCIONES Y REDUCER -----
447 // Exportar acciones sincronas
448 export const {
449   setProductoField,
450   resetProducto,
451   setProductoForEdit,
452   updateMultipleProductosLocal,
453 } = productosSlice.actions;
454
455 // Exportar el reducer para el store
456 export default productosSlice.reducer;
```

4.6.1.8.10 Conclusión Slice

Conclusión institucional

Usar `createSlice + createAsyncThunk` permite:

Modularidad claro entre lógica local y remota

Trazabilidad completa en DevTools

Reproducibilidad y escalabilidad

Código limpio, validado y listo para documentación

Es la arquitectura recomendada por Redux Toolkit para aplicaciones modernas con backend, formularios, y flujos asincrónicos como Excel.

Aguadelo
Gutiérrez

4.6.1.9 Usándolo en el hook Custom

4.6.1.9.1 Importar las acciones y Reducer del slice

```
store.js      productosSlice.js ●  useCustomProductos.js 6 ●  productoService.js ●  pro...
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > ...
1  import { useContext, useEffect, useReducer, useState } from "react";
2  > import { ...
7  } from "../../../../Services/productosService/productoService";
8  import Swal from "sweetalert2";
9  import { ModalsContext } from "../../../../Context/ContextModals/ModalsContext";
10 import { useDispatch, useSelector } from "react-redux"
11
12 // importe del slice Reducers y acciones asincronas
13 import {
14   fetchProductos,
15   createProducto,
16   updateProductoThunk,
17   deleteProductoThunk,
18   importProductosThunk,
19   updateMultipleProductosThunk,
20   setProductoField,
21   resetProducto,
22   setProductoForEdit,
23 } from "../../../../store/slice/productos/productosSlice";
24
25
26 > export const useCustomProductos = () => { ...
771 };
772
```

4.6.1.9.2 Funcion Dispatch y useSelector

```
store.js      productosSlice.js ●  useCustomProductos.js 7 X  productoService.js ●
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > ...
1  import { useContext, useEffect, useReducer, useState } from "react";
2  > import { ...
7  } from "../../../../Services/productosService/productoService";
8  import Swal from "sweetalert2";
9  import { ModalsContext } from "../../../../Context/ContextModals/ModalsContext";
10 import { useDispatch, useSelector } from "react-redux"
11
12 // importe del slice Reducers y acciones asincronas
13 > import { ...
23 } from "../../../../store/slice/productos/productosSlice";
24
25
26 export const useCustomProductos = () => {
27
28   // Importar el contexto para abrir el modal de edición
29   const { openProductModaledit } = useContext(ModalsContext);
30
31   // dispatch: Función para enviar acciones a Redux
32   const dispatch = useDispatch();
33
34   // useSelector: Lee el estado del store de Redux
35   // state.productos viene del nombre definido en configureStore
36   const { productos, producto, loading, error } = useSelector(
37     (state) => state.productos
38   );

```

4.6.1.9.3 Handler Genérico y useEffect cargar los datos

```
store.js productosSlice.js useCustomProductos.js productoService.js productosReducer.js productoValidationSchema.js
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > useCustomProductos

19
20 export const useCustomProductos = () => {
21
22 // ----- HOOKS Y CONTEXTOS -----
23 // -----
24
25 // Contexto para manejar modales
26 > const { openProductModaledit } = useContext(ModalsContext); ...
27 const { productos, producto, loading, error } = useSelector(
28   (state) => state.productos
29 );
30
31 // Estado Local para filtrado de productos
32 const [productoFiltrado, setProductoFiltrado] = useState(null);
33
34 //----- EFECTO: Cargar productos al iniciar-----
35
36 // useEffect se ejecuta cuando el componente se monta Aquí cargamos los productos desde el backend
37
38 useEffect(() => {
39   console.log(" Iniciando carga de productos...");
40   dispatch(fetchProductos()); // Despacha el thunk para obtener productos
41   [dispatch]; // Solo se ejecuta una vez al montar el componente
42
43 // ----- HANDLERS DE FORMULARIO-----
44
45 // Handler genérico para actualizar campos del formulario
46 const handleField = (field) => (e) => {
47   // Determinar el valor según el tipo de input
48   const value = e.target.type === "checkbox"
49     ? e.target.checked // Para checkboxes, usar checked (booleano)
50     : e.target.value; // Para otros inputs, usar value (string/number)
51
52   console.log(` Actualizando campo ${field}: ${value}`);
53
54   // Despachar acción para actualizar el campo en Redux
55   dispatch(setProductoField({ field, value }));
56 };
57
58 // Resetear formulario a valores iniciales
59
60 const resetFormulario = () => {
61   console.log(" Reseteando formulario...");
62   dispatch(resetProducto());
63 };
64
65
66
67
68
69
70
71
```

Esto ya se vio en su momento. (Revisar más atrás en la guía)

Gutiérrez

4.6.1.9.4 Crud Agregar en el hook Custom (Thunk)

```
store.js      productosSlice.js ● useCustomProductos.js ● productoService.js ● productosReducer.js
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > [useCustomProductos.js]
21  export const useCustomProductos = () => {
22    ...
23    ...
24    ...
25    ...
26    ...
27    ...
28    ...
29    ...
30    ...
31    ...
32    ...
33    ...
34    ...
35    ...
36    ...
37    ...
38    ...
39    ...
40    ...
41    ...
42    ...
43    ...
44    ...
45    ...
46    ...
47    ...
48    ...
49    ...
50    ...
51    ...
52    ...
53    ...
54    ...
55    ...
56    ...
57    ...
58    ...
59    ...
60    ...
61    ...
62    ...
63    ...
64    ...
65    ...
66    ...
67    ...
68    ...
69    ...
70    ...
71    ...
72    ...
73    ...
74    ...
75    ...
76    ...
77    ...
78    ...
79    ...
80    ...
81    ...
82    ...
83    ...
84    ...
85    ...
86    ...
87    ...
88    ...
89    ...
90    ...
91    ...
92    ...
93    ...
94    ...
95    ...
96    ...
97    ...
98    ...
99    ...
100   ...
101   ...
102   ...
103   ...
104   ...
105   ...
106   ...
107   ...
108   ...
109   ...
110   ...
111   ...
112   ...
113   ...
114   ...
115   ...
116   ...
117   ...
118   ...
119   ...
120   ...
121   ...
```

```
const handlerAddproducto = async (producto) => {
```

Define una función asíncrona llamada `handlerAddproducto`.

Recibe como parámetro el objeto `producto` con los datos que se van a enviar al backend.

Se usa `async` porque internamente se espera una operación asíncrona (thunk).

```
try {
```

Inicia el bloque de manejo de errores.

Si ocurre una excepción durante la creación, se captura en el bloque `catch`.

```
console.log("Agregando producto:", producto);
```

Log institucional para trazabilidad.

Muestra en consola el objeto `producto` que se va a despachar.

```
await dispatch(createProducto(producto)).unwrap();
```

- ◆ Despacha el thunk `createProducto` con el producto como argumento.
- ◆ `dispatch()` envía la acción al store de Redux.
- ◆ `.unwrap()` convierte la promesa del thunk en una promesa estándar:
 - Si el thunk se resuelve (`fulfilled`), retorna el payload.
 - Si falla (`rejected`), lanza una excepción que puede ser capturada por el `catch`.
- ◆ Esto permite manejar el resultado como si fuera una promesa normal, sin necesidad de usar

4.6.1.9.5 Función onSubmit

```
store.js      productosSlice.js ● useCustomProductos.js ● productoService.js ● productosReducer.js
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > [o] useCustomProductos.js
21  export const useCustomProductos = () => {
22    ...
23
24  >  const resetFormulario = () => { ...
25
26  >  //----- UTILIDADES -----
27  >  const getImagenesProducto = () => { ...
28
29
30  >  //----- CREATE: Agregar producto-----
31  >  const handlerAddproducto = async (producto) => { ...
32
33
34
35  >  const onSubmit = async (e) => {
36    e.preventDefault(); // Prevenir recarga de página
37
38    console.log(" Enviando formulario...");
39
40    // Validación: Verificar que se seleccionó una subcategoría
41    if (!producto.id_subCategorias) {
42      Swal.fire("Error", "Debe seleccionar una subcategoría", "error");
43      return;
44    }
45
46    // Intentar agregar el producto
47    const result = await handlerAddproducto(producto);
48
49    // Si se agregó exitosamente, limpiar el formulario
50    if (result) {
51      console.log(" Producto agregado, limpiando formulario...");
52      dispatch(resetProducto());
53    }
54
55  };
56
```

4.6.1.9.6 handleEdit (prepara la edición.)

```
store.js      productosSlice.js ● useCustomProductos.js ● productoService.js ● produ
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > [o]
21  export const useCustomProductos = () => {
90
91
92  //----- CREATE: Agregar producto-----
93  > const handlerAddproducto = async (producto) => {
121  };
122
123
124 > const onSubmit = async (e) => {
143  };
144 // -----UPDATE: Actualizar producto-----
145
146 // Cargar producto para edición
147 const handleEdit = (productoToEdit) => {
148   console.log("📝 Cargando producto para editar:", productoToEdit);
149
150 // Cargar datos en el formulario
151 dispatch(setProductoForEdit(productoToEdit));
152
153 // Abrir modal de edición
154 openProductModaledit();
155
156
157 // Actualizar producto (compatible con Formik)
158 > const handleUpdateProduct = async (values, formikHelpers) => {
198
199
```

Esto ya se vio en su momento. (Revisar más atrás en la guía.

◆ `handleEdit(productoToEdit)`

Esta función se ejecuta **cuando el usuario hace clic en "Editar"** sobre un producto.

- **Carga los datos** del producto en el estado global (`dispatch(setProductoForEdit())`).
- **Abre el modal** de edición (`openProductModaledit()`).
- No realiza ninguna operación con el backend.
- Su único propósito es preparar la UI para que el usuario pueda editar.
- ♦ Es parte del flujo **visual y de preparación**.

4.6.1.9.7 Crud Editar (handleUpdateProduct) Enviar al backend los datos

```
store.js          productosSlice.js • useCustomProductos.js • productoService.js • productosRed
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > useCustomProductos.js > useCustomProductos.js
21   export const useCustomProductos = () => {
147 >   const handleEdit = (productoToEdit) => { ...
155   };
156
157   // Actualizar producto (compatible con Formik)
158   const handleUpdateProduct = async (values, formikHelpers) => {
159     const { setSubmitting, setErrors } = formikHelpers;
160
161   try {
162     console.log(" Actualizando producto:", values);
163
164     // Despachar thunk y esperar resultado
165     await dispatch(updateProductoThunk(values)).unwrap();
166
167     // Mostrar mensaje de éxito con temporizador
168     await Swal.fire({
169       icon: "success",
170       title: "Producto Actualizado",
171       text: `Producto "${values.nombre}" actualizado con éxito!`,
172       timer: 2000,
173       showConfirmButton: false,
174     });
175
176     console.log(" Producto actualizado exitosamente");
177     return true;
178
179   } catch (error) {
180     console.error(" Error al actualizar producto:", error);
181
182     // Usar setErrors de Formik para mostrar el error en el formulario
183     // Esto permite que Formik maneje el error y lo muestre en el componente
184     setErrors({
185       submit: error, // Campo especial para errores generales
186     });
187
188     // Mostrar alerta de error
189     Swal.fire("Error", error || "Error al actualizar", "error");
190
191     return false;
192
193   } finally {
194     // IMPORTANTE: Siempre deshabilitar el estado de "enviando"
195     // Esto reactiva el botón de submit después de la operación
196     setSubmitting(false);
197   }
198 };
199
```

```
const handleUpdateProduct = async (values, formikHelpers) => {
```

- ◆ Define una función asíncrona llamada `handleUpdateProduct`.
- ◆ Recibe dos parámetros:
 - `values` : objeto con los datos del formulario.
 - `formikHelpers` : objeto con funciones auxiliares de Formik (`setSubmitting`, `setErrors`).

```
const { setSubmitting, setErrors } = formikHelpers;
```

- Extrae las funciones necesarias para controlar el estado del formulario:

`setSubmitting` : activa/desactiva el estado de envío.

`setErrors` : permite mostrar errores en el formulario.

```
try {
```

- ◆ Inicia el bloque de manejo de errores.

```
  console.log(" Actualizando producto:", values);
```

Log institucional para trazabilidad.

Muestra los datos que se van a enviar al backend.

```
  await dispatch(updateProductoThunk(values)).unwrap();
```

- ◆ Despacha el thunk `updateProductoThunk` con los datos del formulario.

- ◆ `.unwrap()` convierte la promesa del thunk en una promesa estándar:

- ◆ Si se resuelve, continúa.

- ◆ Si falla, lanza una excepción que se captura en el `catch`.

```
  await Swal.fire({
    icon: "success",
    title: "Producto Actualizado",
    text: `Producto "${values.nombre}" actualizado con éxito!`,
    timer: 2000,
    showConfirmButton: false,
});
```

- ◆ Muestra una alerta de éxito con temporizador.

- ◆ No requiere confirmación del usuario.

- ◆ Refuerza la experiencia visual del flujo exitoso.

```
  console.log(" Producto actualizado exitosamente");
```

- ◆ Log de confirmación técnica.

- ◆ Útil para auditoría y debugging.

```
  return true;
```

- ◆ Retorna `true` para indicar que la operación fue exitosa.

Js ^

```
} catch (error) {
```

- ◆ Captura cualquier error ocurrido durante la actualización.

```
  console.error(" Error al actualizar producto:", error);
```

- ◆ Log de error para diagnóstico técnico.

```
setErrors({  
  submit: error, // Campo especial para errores generales  
});
```

- ◆ Usa `setErrors` de Formik para mostrar el error en el formulario.
- ◆ El campo `submit` es una convención para errores generales no ligados a campos específicos.

```
Swal.fire("Error", error || "Error al actualizar", "error");
```

- ◆ Muestra una alerta visual de error.
- ◆ Refuerza la retroalimentación al usuario.

```
return false;
```

- ◆ Retorna `false` para indicar que la operación falló.

```
} finally {
```

- ◆ Se ejecuta siempre, haya éxito o error.

Elemento	Propósito
<code>dispatch(updateProductoThunk)</code>	Ejecuta la actualización en backend
<code>.unwrap()</code>	Manejo estándar de promesas
<code>Swal.fire()</code>	Feedback visual
<code>setErrors()</code>	Mostrar errores en el formulario
<code>setSubmitting(false)</code>	Reactivar botón de envío

- ◆ Todo encapsulado en una función modular, compatible con Formik y trazable.

4.6.1.9.8 Crud Eliminar (handleDelete) Eliminar info

```
js store.js      js productosSlice.js ● js useCustomProductos.js X js productoService.js ● js productosRe
redux > mayordomo > src > Pages > Anmind > hooks > hooksCustom > Productos > js useCustomProductos.js > [x] useCu
21  export const useCustomProductos = () => {
156
157    // Actualizar producto (compatible con Formik)
158  >  const handleUpdateProduct = async (values, formikHelpers) => { ...
198
199
200 // ----- DELETE: Eliminar producto-----
201 >  const handleDelete = (id) => {
202   >  console.log("Solicitando eliminar producto ID:", id);
203
204   // Mostrar diálogo de confirmación
205   Swal.fire({
206     title: "¿Estás seguro?",
207     text: "¡No podrás revertir esto!",
208     icon: "warning",
209     showCancelButton: true,
210     confirmButtonColor: "#3085d6",
211     cancelButtonColor: "#d33",
212     confirmButtonText: "Sí, eliminar!",
213     cancelButtonText: "Cancelar",
214   }).then(async (result) => {
215     // Si el usuario confirma
216     if (result.isConfirmed) {
217       >  try {
218         >  console.log("Eliminando producto...");
219
220         // Despachar thunk de eliminación
221         await dispatch(deleteProductoThunk(id)).unwrap();
222
223         // Mostrar mensaje de éxito
224         Swal.fire(
225           "Eliminado!",
226           "Producto eliminado con éxito.",
227           "success"
228         );
229
230         console.log("Producto eliminado exitosamente");
231       } catch (error) {
232         console.error("Error al eliminar:", error);
233
234         // Mostrar mensaje de error
235         Swal.fire("Error", error || "Error al eliminar", "error");
236       }
237     } else {
238       >  console.log("Eliminación cancelada por el usuario");
239     }
240   });
241};
```

5 qué es Reselect?

es una librería que permite crear **selectores memoizados** para Redux. Un selector es una función que lee y transforma el estado global. Un selector memoizado evita cálculos innecesarios si el estado no ha cambiado.

5.1 Ventajas institucionales:

- Evita re-renderizados innecesarios
- Mejora el rendimiento en componentes conectados
- Permite derivar datos complejos sin recalcular
- Compatible con Redux Toolkit y useSelector.

5.2 ¿Cuándo utilizar Reselect?

Situación	Ejemplo
Filtras datos	<code>productos.filter(p => p.activo)</code>
Calculas valores	<code>productos.reduce(...)</code>
Agrupa o transformas	<code>groupBy(productos, 'categoria')</code>
El componente se re-renderiza sin necesidad	<code>useSelector</code> recalcula aunque no cambie el resultado

5.3 Cuando no utilizarlo:

No lo uses para leer valores simples (`state.loading`, `state.error`)

Úsalo solo si haces transformaciones (filtrar, agrupar, calcular)

Ubícalo en carpeta `selectors/` o dentro del slice si es simple

Memoiza solo si hay beneficio real (evita sobreingeniería)