Explicación UT2A3 — Páginas 16–20 (Redux Toolkit)

Objetivo: entender y aplicar la **gestión global de estado** con **Redux Toolkit**

Antes de empezar: instalación

npm install @reduxjs/toolkit react-redux
npm install --save-dev @types/react-redux

1. Introducción: por qué Redux Toolkit

• Problema a resolver: cuando un dato (p. ej., usuario/rol) debe consultarse en varias páginas, pasarlo por props es incómodo y frágil.
• Solución: crear un **almacén global de estado (store)** accesible desde cualquier componente.
• Con Redux Toolkit: definimos **slices** (estado + acciones) y un **store** centralizado.

¿Qué es un *slice* en Redux Toolkit?

Un **slice** (en inglés, "rebanada") es una **parte del estado global** de la aplicación, junto con la **lógica necesaria para modificarlo**.

Piensa en la aplicación como una **tarta de datos (el estado completo)**.
Cada *slice* es una **porción de esa tarta**, que se encarga de un tipo concreto de información.

- Un *slice* para la autenticación del usuario (authSlice).
- Otro *slice* para productos (productosSlice).
- Otro *slice* para configuración (configSlice).

Cada slice contiene:

1. **Un estado inicial** (por ejemplo: usuario no autenticado).
2. **Reducers**, que son funciones que explican **cómo cambia ese estado**.
3. **Acciones**, que son comandos que lanzan esos cambios.

Ejemplo:

```
// authSlice.ts
import { createSlice } from '@reduxjs/toolkit'

// 1️⃣ Estado inicial
const estadoInicial = {
  autenticado: false,
  nombreUsuario: '',
  rol: ''
}

// 2️⃣ Creamos el slice
const authSlice = createSlice({
  name: 'autenticacion',   // nombre de la "rebanada"
  initialState: estadoInicial,
  reducers: {
    // Funciones que describen cómo cambia el estado
    login: (state, action) => {
      state.autenticado = true
      state.nombreUsuario = action.payload.nombre
      state.rol = action.payload.rol
    },
```

```
  logout: (state) => {
    state.autenticado = false
    state.nombreUsuario = ''
    state.rol = ''
  }
 }
})

// 3️⃣ Exportamos las acciones y el reducer
export const { login, logout } = authSlice.actions
export default authSlice.reducer
```

¿Qué es el *store*?

El **store** (almacén) es el **contenedor central** que guarda **todos los slices** de tu aplicación. Es decir, si cada slice es una "rebanada de datos", el store es **el plato donde las juntas todas**.

El store:

- **Guarda el estado global completo.**
- **Escucha las acciones** que se envían desde los componentes (por ejemplo, login, logout, añadir producto…).
- **Distribuye los cambios** a las partes que los necesitan.

Ejemplo:

```
// store/index.ts
import { configureStore } from '@reduxjs/toolkit'
import authReducer from './authSlice'  // importamos el slice de autenticación

// Creamos el store y le decimos qué slices contiene
export const store = configureStore({
```

```
 reducer: {
   autenticacion: authReducer   // “autenticacion” es una clave dentro del estado global
 }
})


// Tipos útiles para TypeScript
export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch
```

Cómo se usan juntos en la aplicación

1. En main.tsx, **envolvemos la app** con el <Provider store={store}> para que todos los componentes puedan acceder al estado.
2. En los componentes:
    - **Leemos** datos del store con useSelector().
    - **Cambiamos** datos del store con useDispatch() enviando acciones del slice.

```
// 1️⃣ En Login: cuando el usuario entra correctamente
dispatch(login({ nombre: 'María', rol: 'admin' }))


// 2️⃣ En Home: mostramos datos del store
const usuario = useSelector((state: RootState) => state.autenticacion)
console.log(usuario.nombreUsuario, usuario.rol)
```

Concepto	**Qué representa**	**Lo nuevo que aporta**
Slice	Porción del estado global (datos + funciones)	Modulariza la lógica y evita mezclarla con la interfaz
Store	Almacén central donde viven todos los slices	Permite compartir datos entre páginas o componentes
Reducers	Funciones dentro del slice	Indican cómo cambia el estado

Concepto	Qué representa	Lo nuevo que aporta
Acciones (actions)	"Órdenes" que activan un cambio en el estado	Se ejecutan con dispatch()
Hooks	useDispatch y useSelector	Permiten modificar y leer el estado desde cualquier componente

2. Estructura de carpetas/archivos (recomendada)

• src/store/index.ts → Configura el store.
• src/store/authSlice.ts → Define el slice de autenticación (estado + reducers/acciones).
• src/paginas/Login.tsx → Usará dispatch para iniciar/cerrar sesión.
• src/paginas/Home.tsx → Usará selector para leer usuario/rol desde el store.
• src/main.tsx → Envolverá la app con <Provider store={store}>.

3. Creación del STORE (configureStore) — src/store/index.ts

```
// src/store/index.ts — Configuración del store global con Redux Toolkit
import { configureStore } from '@reduxjs/toolkit'   // Importamos la función para crear el store
// import authReducer from './authSlice'          // (Se importará después de crear el slice)

// Creamos el store con un objeto reducer que agrupa los slices de la app
export const store = configureStore({            // 'store' es el almacén global
 reducer: {                                      // Aquí registraremos nuestros slices
   // autenticador: authReducer                  // <- Lo activaremos cuando creemos el slice
 },
})

// Tipos derivados del propio store para TypeScript
export type RootState = ReturnType<typeof store.getState>  // Tipo del estado raíz (para useSelector)
export type AppDispatch = typeof store.dispatch          // Tipo de dispatch (para useDispatch)
```

4. Creación de un SLICE (createSlice) — src/store/authSlice.ts — COMENTADO

```
// src/store/authSlice.ts — Slice de autenticación (usuario + rol + estado)
// Un slice define: nombre, estado inicial y reducers (acciones que modifican el estado)

import { createSlice, PayloadAction } from '@reduxjs/toolkit' // Herramientas de Redux Toolkit

// 1) Definimos la forma del estado (TypeScript)


export interface EstadoAuth {
  autenticado: boolean       // ¿Hay sesión iniciada?
  nombreUsuario: string      // Nombre a mostrar
  rolUsuario: string         // Rol (p. ej., 'admin', 'editor', etc.)
}

// 2) Estado inicial del slice
const estadoInicial: EstadoAuth = {
  autenticado: false,        // Por defecto no hay sesión
  nombreUsuario: '',         // Sin nombre
  rolUsuario: ''             // Sin rol
}

// 3) Creamos el slice con nombre, estado inicial y reducers
const authSlice = createSlice({
  name: 'autenticacion',     // Nombre interno del slice
  initialState: estadoInicial,
  reducers: {
    // Acción 'login': activa sesión y guarda datos básicos
    login: (
      state,
      action: PayloadAction<{ nombre: string; rol: string }>
    ) => {
      state.autenticado = true               // Marcamos sesión activa
      state.nombreUsuario = action.payload.nombre  // Guardamos nombre
      state.rolUsuario = action.payload.rol        // Guardamos rol
    },
    // Acción 'logout': limpia completamente los datos de sesión
    logout: (state) => {
      state.autenticado = false
      state.nombreUsuario = ''
```

```
    state.rolUsuario = ''
   }
  }
})

// 4) Exportamos las acciones y el reducer
export const { login, logout } = authSlice.actions  // { login, logout } para dispatch
export default authSlice.reducer                    // Reducer que se registra en el store
```

5. Registrar el slice en el STORE — src/store/index.ts

```
// src/store/index.ts — Ahora importamos y registramos el reducer del slice
import { configureStore } from '@reduxjs/toolkit'
import authReducer from './authSlice'          // Importamos el reducer del slice de auth

export const store = configureStore({
  reducer: {                                   // Objeto con todos los reducers de la app
    autenticador: authReducer,                 // Registramos nuestro slice con una clave
  },
})

// Tipos auxiliares para TS (igual que antes)
export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch
```

6. Integrar el STORE en la App con <Provider> — src/main.tsx

```
// src/main.tsx — Envolvemos la app con <Provider store={store}>
import React from 'react'                      // React para JSX
import ReactDOM from 'react-dom/client'        // Render en el DOM
import App from './App'                        // Componente raíz
import { Provider } from 'react-redux'         // Proveedor de Redux
import { store } from './store'                // Nuestro store configurado

// (Opcional) Si usas MUI para el tema global:
import CssBaseline from '@mui/material/CssBaseline'
import ThemeProvider from '@mui/material/styles/ThemeProvider'
import createTheme from '@mui/material/styles/createTheme'
```

```
const tema = createTheme({                    // Tema básico de ejemplo
  palette: { primary: { main: '#0B5AA2' } }
})

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>                          {/* Modo estricto en desarrollo */}
    <ThemeProvider theme={tema}>              {/* Tema global (opcional) */}
      <CssBaseline />                         {/* Normaliza estilos */}
      <Provider store={store}>                {/* Proveedor del STORE Redux */}
        <App />                               {/* Toda la app con acceso al store */}
      </Provider>
    </ThemeProvider>
  </React.StrictMode>
)
```

7. Usar Redux: dispatch y selector — COMENTADO

7.1. Enviar acciones (login/logout) desde una página de Acceso/Login:

```
// src/paginas/Acceso.tsx — Ejemplo de uso de dispatch para iniciar/cerrar sesión
import React from 'react'
import { useDispatch } from 'react-redux'       // Hook para enviar acciones
import { login, logout } from '../store/authSlice' // Acciones del slice

export default function Acceso(){
  const dispatch = useDispatch()                // Obtenemos la función dispatch

  const manejarAcceso = () => {                 // Simulación de validación OK
    dispatch(login({ nombre: 'Patri', rol: 'docente' })) // Guardamos datos en el store
  }

  const manejarSalida = () => {                 // Cerrar sesión
    dispatch(logout())                          // Limpiamos el estado global de auth
  }

  return (
    <div>
      <button onClick={manejarAcceso}>Entrar</button>
```

```
    <button onClick={manejarSalida}>Salir</button>
   </div>
  )
}
```

7.2. Leer el estado global (usuario/rol) en otra página, con useSelector:

```
// src/paginas/Inicio.tsx — Ejemplo de lectura del estado global
import React from 'react'
import { useSelector } from 'react-redux'       // Hook para leer el estado global
import { RootState } from '../store'            // Tipo del estado raíz

export default function Inicio(){
  // Leemos la rama 'autenticador' que registramos en el store
  const { autenticado, nombreUsuario, rolUsuario } = useSelector(
    (state: RootState) => state.autenticador
  )

  return (
    <div>
      {autenticado ? (
        <p>Hola, {nombreUsuario} ({rolUsuario})</p>
      ) : (
        <p>No has iniciado sesión.</p>
      )}
    </div>
  )
}
```

```
// • El enrutador (react-router-dom) sigue igual, pero ahora las páginas pueden
//   consultar el estado global (ej.: si 'autenticado' es true, mostrar Panel).
// • El tema MUI (ThemeProvider + CssBaseline) convive con Redux sin interferir.
// • Recomendación: proteger rutas privadas comprobando 'autenticado' desde el store.
```

9. Resumen final

1) Crear un store con configureStore y derivar los tipos RootState y AppDispatch.
2) Crear un slice con createSlice (estado inicial + reducers/acciones).
3) Registrar el reducer del slice en el store.
4) Envolver <App/> con <Provider store={store}> en main.tsx.
5) Usar useDispatch para enviar acciones y useSelector para leer el estado desde cualquier página.
6) Mantener la separación de responsabilidades: tema (MUI), rutas (router) y estado (Redux).