Actividad evaluable UT2A3 (AE1.3) – Redux

RA1, RA3, RA4

En esta actividad continuaremos con el proyecto de UT2. En esta ocasión aprenderán a pasar datos del usuario entre la página Login y otras.

¿Qué tengo que entregar? (Requisito)

- Enlace al github en la rama redux.
- Memoria en formato PDF con capturas de pantalla y breve explicación de lo especificado en verde en este documento. La memoria debe tener portada e índice, el texto debe tener alineación justificada, la expresión escrita debe ser correcta. Numera las páginas de la memoria.
- El proyecto será descargado por la profesora desde el github: es requisito indispensable que la aplicación se pueda ejecutar en cualquier equipo para poder ser corregida.
- No se admite código generado por IA (y sí, se nota ...)

¿Cómo identifico mi trabajo? (Requisito)

- Debes poner tu nombre y apellidos en el fichero README.md
- Debes nombrar el PDF como sigue:

PrimerApellido_SegundoApellido_Nombre_UT2A3

Desarrollo de la actividad

1. Creación de una nueva rama

Crea una nueva rama en tu proyecto: te sitúas en la carpeta proyecto*Tusiniciales* y ahí escribes el siguiente comando:

```
git checkout -b redux
```

Con esto estás creando una rama nueva llamada redux y además, te estás cambiando a esa rama.

Comprueba que estás en la nueva rama con el comando:

```
git branch
```

2. Explicación de redux (necesario leer esto para entender cómo implementarlo)

A veces, necesitaremos pasar estados de los componentes entre una página y otra. Esto se puede hacer con los props, pero su gestión puede resultar engorrosa. Una forma de facilitar el trabajo es usando librerías tipo redux. En el siguiente vídeo se explica el problema y por qué usar librerías redux: **https://www.youtube.com/watch?v=j-jzI3wkkVk**

En resumen, redux nos sirve para almacenar estados de componentes en una especie de base de datos en la cual nosotros ponemos estados, los consultamos cuando lo necesitemos y lo cambiamos cuando sea necesario.

2.1 ¿Por qué necesitamos el redux en nuestra aplicación?

En nuestra aplicación tenemos un login en el que si el usuario se autentica correctamente pasará a la página /home.

En la página /home usaremos el nombre del usuario para mostrarlo. Ejemplos: al ingresar en github se muestra tu nombre de usuario, lo mismo al ingresar en cualquier red social, ...

Además, en nuestra aplicación podrán tener acceso varios usuarios con diferentes roles. Por ejemplo, puede haber un usuario con el rol de administrador que tenga todos los permisos y otro usuario con el rol de usuario estándar que tenga menos permisos que el administrador. Por lo tanto, debemos saber qué rol tiene el usuario que ingresa a nuestra aplicación y ese dato tiene que estar disponible en la página /home.

En resumen, **queremos pasar el nombre de usuario y el rol de usuario a las diferentes páginas de nuestra aplicación**. Por defecto, **cuando pasamos de la página de login a cualquier otra página, esta información se perderá. Con el redux la podemos almacenar y por tanto tenerla disponible**.

2.3 Explicación del funcionamiento del redux toolkit

Con el redux podemos crear un store (almacenamiento) de los estados de nuestra aplicación. Estos estados se pueden cambiar usando el useDispatch y se pueden obtener (o consultar) usando el useSelector, ambos son *hooks* de la librería de redux-react.

Para crear el store usaremos 2 funciones de la librería de redux-toolkit:

configureStore(): crea el store y usa los llamados slice reducers, que son funciones que nos dicen el estado y la acción para cambiar dicho estado.
createSlice(): es donde creamos la lógica de nuestra store. Tiene un nombre, un estado inicial y las funciones reducers que representan los estados y acciones.

→ Para **cambiar el estado en el store** usamos el hook: **useDispatch()**
→ Para **obtener el estado del store** usamos el hook: **useSelector()**

3. Implementación del redux en nuestra aplicación

En la siguiente referencia tienen un ejemplo completo de cómo usar redux-toolkit en una aplicación de React. Yo me he basado en ese ejemplo y lo he personalizado para nuestra aplicación.
Referencia: **https://redux-toolkit.js.org/tutorials/quick-start**

3.1. Instalación de las librerías necesarias en el frontend: librería del redux-toolkit y librería del redux de react. Nos situamos dentro de la carpeta frontend de nuestra aplicación:

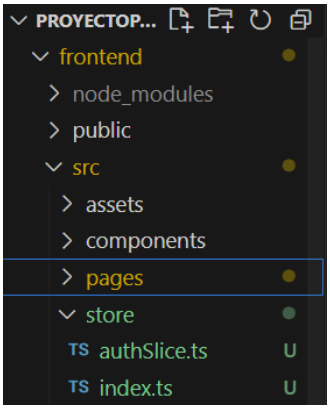
```
cd frontend
npm install @reduxjs/toolkit
npm install react-redux
```

3.2. Creamos una carpeta dentro de **frontend/src** que se llame store, para programar ahí la parte del redux:

```
cd src
mkdir store
```

3.3. Dentro de la carpeta store crearemos los siguientes dos ficheros (botón derecho encima de la carpeta store, New File): index.ts y authSlice.ts

La estructura de carpetas y ficheros debe quedar como la siguiente:



3.4. Abrimos el fichero **frontend/src/store/index.ts** para crear el store de redux con la función configureStore(). Por ahora esto no hace nada, sólo hemos creado el store.

```
import { configureStore } from '@reduxjs/toolkit'

export const store = configureStore({
  reducer: { },
})

// Infer the `RootState` and `AppDispatch` types from the store itself: tipos para typescript
export type RootState = ReturnType<typeof store.getState>
// Inferred type: {posts: PostsState, comments: CommentsState, users: UsersState}: tipos para typescript
export type AppDispatch = typeof store.dispatch
```

3.5. Una vez tengamos creado el store, lo tenemos que hacer disponible para los componentes de React envolviendo nuestra <App> con un <Provider> de React-Redux. Para eso, abrimos nuestro fichero **frontend/src/main.tsx** y añadimos lo siguiente (**OJO, el código que ya tengan puesto, lo deben dejar como está**):

```
//Importamos el componente Provider de la librearía react-redux
import { Provider } from 'react-redux'
//Importamos el componente store que definimos en el fichero ./store/index
import { store } from './store/index'

//Todo el código que tenían de otras importaciones y el código de createTheme lo dejan como está.
//Finalmente escribimos lo siguiente: lo que está en púrpura es lo que añadí a lo que ya estaba.
createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <ThemeProvider theme={customTheme}>
      <CssBaseline />
      <Provider store={store}>
        <App />
      </Provider>
    </ThemeProvider>
  </StrictMode>,
)
```

3.6. Ahora creamos un ***slice*** de redux con la función createSlice. Abrimos el fichero **frontend/src/store/authSlice.ts** y copiamos el siguiente código. Tú puedes cambiar el nombre de las variables, siempre teniendo en cuenta el propósito de las mismas.

```
import { createSlice } from '@reduxjs/toolkit' // Importamos la función createSlice de la librería redux-toolkit

//Creamos el tipo AuthState. Este tipo será un objeto cuyos elementos son un boolean y dos strings.
export interface AuthState {
  isAutenticated: boolean,
  userName: string,
  userRol: string
}

//Declaramos la variable initialAuthState (que es un objeto) y decimos que es del tipo AuthState.
//Inicializamos los elementos de dicha variable:
//El usuario inicialmente no está autenticado (isAutenticated: false)
//El nombre de usuario y su rol inicialmente son cadenas vacías.
const initialAuthState: AuthState = {
  isAutenticated: false,
  userName: '',
  userRol: ''
}

//Creamos los reducers dentro de la función createSlice y los asignamos a la variable authSlice:
//. Primero le damos un nombre
//. Segundo establecemos el estado inicial
//. Tercero creamos los reducers: login y logout
const authSlice = createSlice({
  name: 'authentication',
  initialState: initialAuthState,
  reducers: {
```

```
    //El reducer login: el usuario está autenticado. Ya tenemos el nombre y rol de usuario en action.payload
    login: (state, action) => {
      const userData = action.payload //Obtenemos el nombre y rol de usuario y lo asignamos a la variable userData
      state.isAutenticated = true  //Establecemos a true isAutenticated en la store
      state.userName = userData.name //Es lo mismo que: action.payload.name --> Establecemos el nombre de usuario en la store
      state.userRol = userData.rol //Es lo mismo que: action.payload.rol --> Establecemos el rol de usuario en la store
    },
    //El reducer logout es cuando el usuario no está autenticado. No hay que hacer ninguna action puesto
    //que no recibimos ningún dato del usuario.
    logout: (state) => {
      state = initialAuthState // Cuando el usuario no está autenticado estamos en el estado inicial.
    }
  }
})

//Exportamos las acciones del reducer en la variable authActions
export const authActions = authSlice.actions
//Exportamos el reducer en sí
export default authSlice.reducer
```

Explicación del código anterior

Nosotros queremos almacenar en nuestro store un estado que nos indique si el usuario está autenticado o no, cuál es el nombre de usuario y cuál es su rol. Creamos un estado inicial al que llamo *initialAuthState* en el que esos elementos tomen sus valores iniciales (el usuario no está autenticado y el nombre de usuario y su rol son cadenas vacías).

Con la función createSlice() creamos los dos posibles estados que tendremos, login (autenticado) y no autenticado (logout). Estos son los reducers, que son los diferentes estados que quiero almacenar y manejar con las acciones.

3.7. Ahora añadimos los reducer del Slice que acabamos de crear a nuestro store. Para ello volvemos a abrir el fichero **frontend/src/store/index.ts** y añadimos lo que está resaltado en amarillo:

```
import { configureStore } from '@reduxjs/toolkit'
import authReducer from './authSlice' // Importamos el reducer

export const store = configureStore({
  reducer: {
    authenticator: authReducer, //Aquí configuramos la store con el reducer creado en el Slice
  },
})

// Infer the `RootState` and `AppDispatch` types from the store itself: para los tipos
export type RootState = ReturnType<typeof store.getState>
// Inferred type: {posts: PostsState, comments: CommentsState, users: UsersState}: para los tipos
export type AppDispatch = typeof store.dispatch
```

3.8 Usar el redux en la aplicación:

En principio el store almacena el estado inicial: usuario no autenticado, nombre y rol de usuario a string vacía.

¿Cuándo y cómo cambiamos el estado del store a login?
Cuando el usuario pica en el botón Acceder y ya hemos comprobado que el login y contraseña son los correctos. Esto se debe hacer en el fichero Login.tsx.
Para hacerlo usamos el hook useDispatch(). Así pues, en el fichero frontend/src/pages/Login.tsx añadimos lo siguiente:

```
//Importamos el useDispatch del react-redux
import { useDispatch} from 'react-redux'
//Importamos las acciones que están en el fichero authSlice.ts
import { authActions } from '../store/authSlice';
```

//Justo después de la definición de la función function Login(){ ponemos el hook useDispatch:
```
const dispatch = useDispatch()
```
//Nos dirigimos al trozo de código donde ya hemos comprobado que el usuario y contraseña son
//correctos y añadimos lo siguiente:

```
//aquí pongo el dispatch para cambiar el estado a login en el store del redux
dispatch(authActions.login({
  name: data.user, //data.user es el nombre de usuario que ha ingresado el usuario
  rol: 'administrador'
}))
```

Puedes poner otro rol diferente: 'invitado', 'standard', ...
Fíjate que yo aquí pongo name y rol, puesto que cuando creamos el reducer en el fichero authSlice.ts, pusimos que el action.payload sería name y rol. Te recuerdo el código aquí:

```
reducers: {
  //El reducer login: el usuario está autenticado. Ya tenemos el nombre y rol de usurio en action.payload)
  login: (state, action) => {
    const userData = action.payload //Obtenemos el nombre y rol de usuario y lo asignamos a la variable userData
    state.isAutenticated = true  //Establecemos a true isAutenticated en la store
    state.userName = userData.name //Es lo mismo que: action.payload.name --> Establecemos el nombre de usuario en la store
    state.userRol = userData.rol //Es lo mismo que: action.payload.rol --> Establecemos el rol de usuario en la store
  },
```

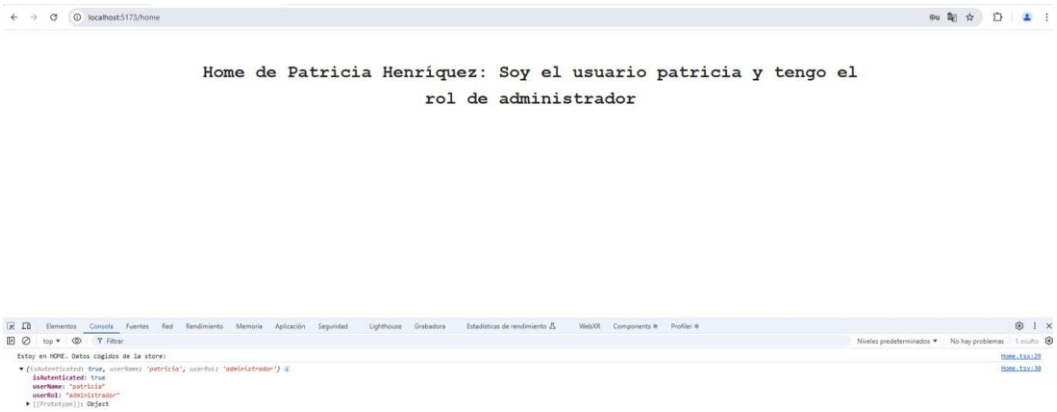
Captura de pantalla del código donde **expliques justificadamente** por qué usas el dispatch y por qué lo pones en el lugar del código que has elegido.

¿Cuándo y cómo obtenemos los datos de login del store?

Cuando entremos en la página /home de nuestra aplicación debemos obtener el nombre y el rol del usuario desde el store. Eso se hace con el hook **useSelector()**. Se hará en el archivo del frontend que maneje la página home, es decir, en **frontend/src/pages/Home.tsx**

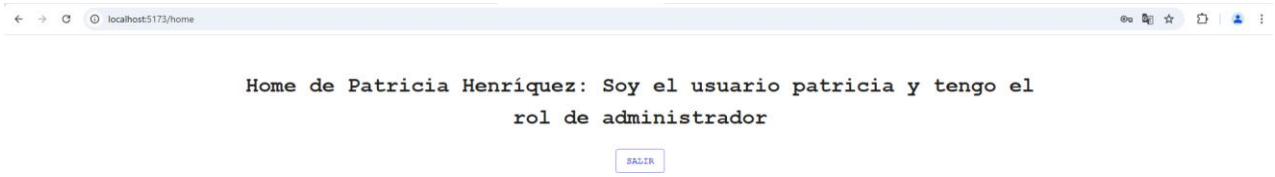
```
//Importamos el useSelector del react-redux
import { useSelector } from 'react-redux'
// Importamos lo que necesitamos para el tipo del selector()
import { RootState} from '../store/index'
//Importamos las acciones que están en el fichero authSlice.ts
import { authActions } from '../store/authSlice';

export default function Home(){
  //Almacenamos en la variable userData lo que obtenemos del store usando el hook useSelector
  const userData = useSelector((state: RootState) => state.authenticator)
  //Comprobamos por la consola qué obtenemos del store
 console.log(userData)
return <>
        //AQUÍ AÑADIREMOS A NUESTRO TYPOGRAPHY EL NOMBRE DE USUARIO Y SU ROL SACADOS DEL STORE.
</>
}
export default Home
```

Implementa el código anterior en Home.tsx y en el Typography añade el nombre de usuario y el rol del usuario que has sacado del store con el useSelector. Además, muestra en la consola los datos sacados del store. Explica el código realizado y saca capturas de pantalla donde se vea el Typography y a la vez la consola.



¿Cuándo y cómo cambiamos el estado del store a logout?

En **frontend/pages/Home.jsx** añadiremos un **botón de SALIR**. En el trozo de código que maneje ese botón tendremos que hacer un dispatch en nuestro store usando el reducer logout y luego navegar a /:



Implementa el botón SALIR con la funcionalidad requerida. Explica justificadamente lo que estás haciendo.

Para ello habrá que importar las librerías necesarias para el useDispatch y el useNavigate.

En el trozo de código que maneje el logout de nuestra aplicación, tendremos que:

1. Hacer un dispatch al store para decirle que estamos en el estado logout: `dispatch(authActions.logout())`
2. Navegar a la página principal de nuestra aplicación, que es /: `navigate('/')`

PUNTUACIÓN

Entrega	Nota MÁXIMA
1) El ejercicio se entrega en plazo y resuelto correctamente.	10
2) Igual que el 1) pero fuera de plazo. Máximo 24 horas	6
3) No se admiten entregas fuera de plazo cuando se han pasado más de 24 horas de la fecha límite de entrega.	0
4) Si no se cumplen las especificaciones de entrega la tarea irá directamente a recuperación.	0

En la tabla se indican las notas máximas en cada situación.