Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México

Ingeniería en Tecnologías computacionales
Diseño y Arquitectura de Software

Selección de patrones a implementar en su proyecto

Autores:
Arturo Efrén Jiménez Garibaldy
Ignacio Alvarado Reyes
José A. Berrón Gutiérrez
Aldo Ponce de la Cruz
Manuel Herrasti González

Profesor:
Marlene Ofelia Sánchez Escobar

Observer

En nuestro proyecto de software uno de los patrones arquitectónicos usados fue Observer. Hacemos uso de este para verificar si existe un carrito de compras para el usuario que tenga la sesión iniciada. En caso de que el usuario agregue un producto al carrito de compras, con el Observer podemos notificar que hubo una actualización en ese componente y debemos de actualizar la información desplegada en la página de la tienda.

Figura 1. Se muestra la diferencia de un carrito de compras vacío y de uno con producto.

Esta implementación de Observer es posible con el uso de la librería React Redux, esta librería permite compartir información entre componentes y hacer cambios cuando uno de estos componentes sufre alguna modificación contemplada en el sistema (como es el ejemplo del cambio en el carrito de compras).

Otra implementación es los cambios que ocurren cuando existe una sesión iniciada. El cambio realizado es mostrar botones o de registro de usuario o inicio de sesión cuando un usuario entra a la pagina sin antes haber usado o creado credenciales, y el otro cambio es cuando ya hay un usuario con una sesión iniciada, es ahí donde mostramos un botón el cual le da la opción de cerrar su sesión actual.

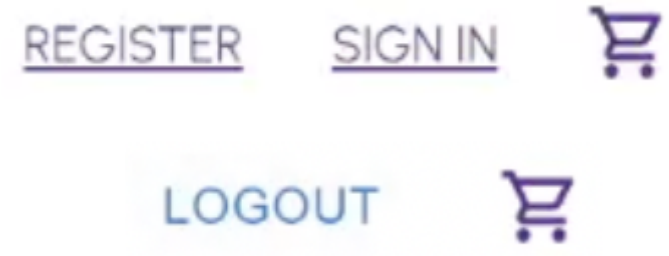

Figura 2. Se muestra la diferencia entre un usuario sin sesión iniciada y un usuario con una sesión iniciada.

Básicamente contamos con un observador que va monitoreando las diferentes operaciones que se realizan en el sistema y este le notifica a los diferentes elementos separados que actúan en la página, cúal es el estado de las variables relacionadas al sistema, como por ejemplo el inicio de sesión de un usuario dónde se mostrarán diferentes contenidos dependiendo si este ha iniciado sesión o no, entonces el observador le avisa a los componentes de react en el ambiente web que hay un estado de sesión y los componentes de react con esta información y su propia funcionalidad ya saben que es lo que tienen que hacer y mostrar.

State

Otro patrón arquitectónico que identificamos en nuestro proyecto es el uso de State, esto principalmente es usado en el proceso de inicio de sesión, cierre de sesión y registro. Los 2 estados que identificamos en nuestro sistema están relacionados al manejo de sesiones en :

- Logged Out: Cuando no se encuentra un usuario con sesión iniciada en la página.
- Logged In: Cuando un usuario está autenticado y activo en la página.

Las acciones que utilizamos para cambiar entre estados son:

- LogInSuccess: En este estado se confirma que el inicio de sesión que se realizó fue exitoso, por ejemplo que los datos como nombre de usuario y contraseña sean correctos, si esta acción se presenta cuando el estado actual en la página es Logged Out, entonces se cambia de estado a Logged In y se hacen las acciones pertinentes.
- LogInFailure: En esta acción notificamos que falló el inicio de sesión por cualquier razón y se cancela la acción que teníamos pensado en caso de que este inicio fuera exitoso. Esto se presentaría cuando se dan datos de inicio de sesión incorrectos cuando la página se encuentra en el estado Logged Out, evitando que se cambie de estado a Logged In y no mostrarle al usuario información a la que no puede acceder.
- LogOut: Esta acción se utilizaría para cerrar la sesión de un usuario cuando se encuentre autenticado en la página, principalmente se utilizaría para pasar del estado Logged In a Logged Out para remover la información que corresponde a cuando un usuario está activo.

Básicamente el patrón de diseño state utiliza una serie de estados que se van intercambiando a través de acciones y cómo estos estados las interpretan para moverse entre sí. Por ejemplo, las transiciones que se realizan entre los diferentes estados se pueden observar de la siguiente manera:

Estado/Acción	LogInSuccess	LogInFailure	LogOut
Logged Out	Logged In	Logged Out	Logged Out
Logged In	Logged In	Logged In	Logged Out

```tsx
const App: React.FC = () => {
  const user = useSelector((state: any) => state.user.currentUser);
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/cart" element={<CartView />} />
        <Route path="/products" element={<ProductListView />} />
        <Route path="/product/:id" element={<ProductView />} />
        <Route
          path="/login"
          element={user ? <Navigate to="/" /> : <LoginView />}
        />
        <Route
          path="/register"
          element={user ? <Navigate to="/" /> : <RegisterView />}
        />
        <Route path="/" element={<HomepageView />} />
      </Routes>
    </BrowserRouter>
  );
};
```

Figura 3. Componente App.tsx principal del front-end. Se muestra la lectura de User state para permitir hacer login o registrarse.

Los states de LogInSucess y LogOut modifican el estado de “User”. Utilizamos la función useSelector de Redux para leer el state de User. Este state es que el permite navegar a los endpoints de “/login” o “/register”. Si el state de user es nulo, entonces React interpreta esto como que no existe un user que ha hecho login y permite visitar estos endpoints.

Otro caso que tenemos es el de Register, el momento cuando el usuario se da de alta dentro de la plataforma y tenemos que ver el estado de nuestro registro en el sistema. Utilizamos los siguientes estados para verificar su funcionamiento:

- RegisterSuccess: En este estado se confirma que la alta de la cuenta fue exitoso, esto se puede deber por ejemplo a que los datos como nombre de usuario y contraseña sean correctos, si esta acción se presenta cuando el estado actual en la página es NotRegistered, entonces se cambia de estado a Registered y se hacen las acciones pertinentes.
- RegisterFailure: En esta acción notificamos que la alta de la cuenta por cualquier razón y se cancela la acción que teníamos pensado en caso de que este registro fuera exitoso. Esto se presentaría cuando los datos de registro de sesión son incorrectos o cuando hubo un error de conexión directa con el servidor.
- UnRegister: Esta acción se utilizaría para borrar la sesión del usuario cuando lo requiera, principalmente se utilizaría para pasar del estado Registered a NotRegistered para remover la información que corresponde a cuando un usuario está registrado.

Básicamente el patrón de diseño state utiliza una serie de estados que se van intercambiando a través de acciones y cómo estos estados las interpretan para moverse entre sí. Por ejemplo,

las transiciones que se realizan entre los diferentes estados se pueden observar de la siguiente manera:

Estado/Acción	RegisterSuccess	RegisterFailure	UnRegister
NotRegistered	Registered	NotRegistered	NotRegistered
Registered	Registered	Registered	NotRegistered

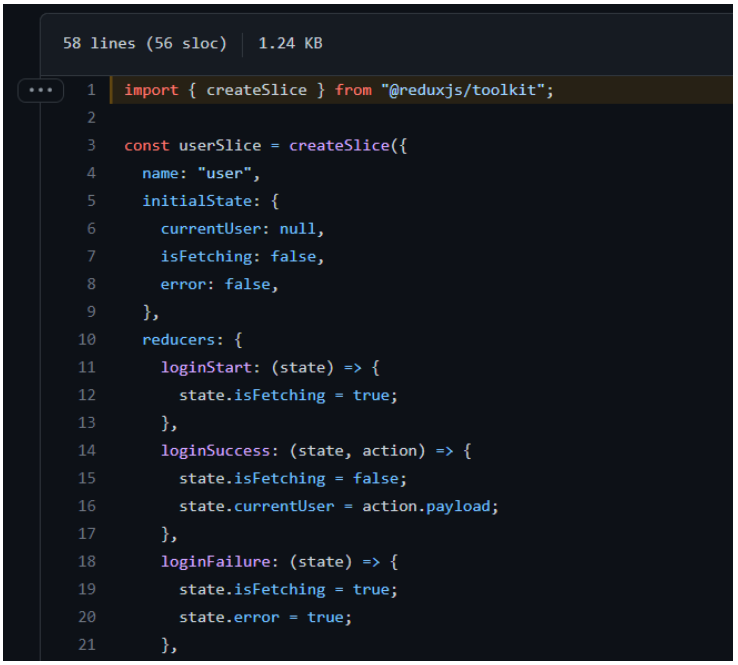

Figura 4. Componente Redux que muestra las funciones que verifican estados de User y sus respectivas acciones.

En esta parte del código mostrado podemos observar que las funciones de Redux reciben como input un estado del usuario y ejecutan una acción. Tanto Login como Register tienen funciones similares. En el caso de registerSuccess, lo que se hace es modificar el estado de currentUser por los datos de usuarios que fueron recibidos desde un forms. Este estado es el que persiste hasta que este haga logout.