

Nodepop Redux

Para esta práctica usaremos de punto de inicio la práctica que se realizó en el modulo de fundamentos. En caso de no disponer de una práctica para empezar, se puede utilizar esta:

<https://github.com/davidij76/nodepop-react-fundamentos>

Objetivos de la práctica:

1. Configurar un store Redux donde se almacenará al menos la siguiente información:
 - a. Información sobre la sesión o el usuario registrado en el sistema. Al iniciar la aplicación se deberá leer la información del usuario desde el LocalStorage (si existiese) y se almacenará en el store de Redux. Al hacer login un usuario su información deberá guardarse en el store de Redux y en el Local Storage si se eligió recordar el login.
 - b. Información sobre los anuncios. El store deberá manejar la obtención de tags disponibles, de anuncios desde el API (listado y detalle), así como la creación y borrado de anuncios.

Será importante modelar correctamente el estado que se va a guardar en el store.

2. Crear las acciones y reducers necesarios para poder cumplir los objetivos del punto 1.
3. Configurar Redux Dev Tools para simplificar las tareas de debugging de la aplicación.
4. Formularios. La aplicación contiene varios formularios (Login, Creación de anuncios). Estaría bien extraer lógica común a todos ellos y reutilizarlos en los distintos formularios, por ejemplo creando un componente `<Form />` que mantenga los valores del formulario y un `<Input />` que reciba el valor que le corresponde así como la función necesaria para poder modificar ese valor en el evento `onChange`. De ese modo, toda la lógica del `onChange` estará “escondida” en los componentes `Form` e `Input`. La idea es que luego en el momento de usar estos componentes se pueda hacer así (ejemplo para el Login).

```
<Form initialValue = {{ email: '', password: '' }} onSubmit = { ... } >
  <Input type = "text" name = "email" />
  <Input type = "password" name = "password" />
</Form>
```

Podeis usar cualquiera de las técnicas que hemos visto en el curso (hoc, hooks, renderProps, context, etc). Con aplicarlo en algún formulario (Login, por ejemplo) será suficiente.

5. Refactorizar algún componente para que use hooks, por ejemplo gestionando su estado con `useState` o sus efectos con `useEffect`.
6. Testing. Crear tests unitarios, dando al menos un ejemplo de cada uno de estos casos.
 - a. Una acción síncrona.
 - b. Una acción asíncrona.
 - c. Un reducer.
 - d. Un selector.
 - e. Un componente con snapshot testing.

- f. Comprobar el funcionamiento de un componente que ejecuta una acción del store, mockeando la acción.