Acciones y Reducers en nuestro proyecto

El primer paso es instalar Redux y React-Redux. Ambas librerías se distribuyen como paquetes `npm` .

```
npm install --save redux react-redux
```

Como hemos visto en el ejemplo de Redux (Codepen), para crear el `Store` primero necesitamos definir el reducer de nuestra aplicación. Las acciones las podemos definirlas más adelante, pero os recomiendo hacerlo antes del reducer. De esta manera ya tendréis en mente los tipos y los parámetros que recibirá el reducer.

Dicho esto, vamos a crear nuestras acciones. Para mantenerlas organizadas, creamos la carpeta `src/actions` . Dentro de esta carpeta definimos nuestras acciones en el fichero actions.js.

- `startSearch(search)` : almacena la búsqueda actual y actualiza `loading` a `true`
- `successSearch(results)` : actualiza `loading` y `queried` y almacena los resultados

El siguiente paso es crear nuestro reducer. Al igual que para las acciones, creamos la carpeta `src/reducers` . En esta carpeta definimos nuestro reducer en el fichero reducer.js. El estado inicial debe contener los valores que hasta ahora se almacenaban en el estado de `SearchContainer.js` :

```
// Estado inicial de nuestra aplicación
const initialState = {
  search: '',
  loading: false,
  results: [],
  queried: false
}
```

Basándonos en las acciones anteriores, definimos nuestro reducer:

```
/**
 * El reducer recibe el estado actual y la acción a ejecutar. Si el estado
 * no está definido, obtenemos el estado por defecto
 */
const reducer = (state = initialState, action) => {
  switch (action.type) {
    /**
     * Comenzamos la búsqueda
     */
    case 'SEARCH_START': {
      return Object.assign({}, state, { loading: true, search: action.search });
    }
```

```
    /**
     * La búsqueda ha terminado.
     */
    case 'SEARCH_SUCCESS': {
      return Object.assign({}, state, {
        loading: false, results: action.results, queried: true
      });
    }
    default: {
      // Es importante retornar por defecto el estado.
      // Si no retornamos nada en un Reducer, el estado se pierde
      return state;

    }
  }
}
```

Ya podemos definir nuestro `store`. Creamos el fichero `src/store.js` que creara el store a través del método `createStore` de la librería.

```
// Importamos el método para crear el store de redux
import { createStore } from 'redux';

// Importamos el reducer de nuestra aplicación
import Reducer from './reducers/reducer';

// Creamos el reducer
const store = createStore(Reducer);
export default store;
```

Una vez definidas las acciones, el reducer y creado el store, es el momento de integrar Redux con nuestro proyecto. La integración es muy similar a React-Router. En este caso, vamos a utilizar el HOC `Provider` que recibe como parámetro el `store` y lo integra en nuestra aplicación. Modificamos el fichero src/index.js:

```
// Styles
import './index.css';

// Importamos las distintas librerias
import React from 'react';
import ReactDOM from 'react-dom';

// Redux
import { Provider } from 'react-redux';
import store from './store';

// Importamos los componentes
import { Router, Route, hashHistory } from 'react-router';
import BaseContainer from './containers/BaseContainer';
import DetailsContainer from './containers/DetailsContainer';
import About from './components/About';

ReactDOM.render(
  <Provider store={ store }>
    <Router history={ hashHistory }>
```

```
    <Router history={ hashHistory }>
      <Route path="/" component={ BaseContainer }>
        <Route path=":user/:repo" component={ DetailsContainer }/>
        <Route path="/about" component={ About }/>
      </Route>
    </Router>
  </Provider>,
  document.getElementById('root')
);
```

Por último, modificamos el fichero src/containers/SearchContainer/SearchContainer.js para hacer uso del estado de Redux. El HOC `connect` nos permite pasar al componente atributos del estado como `props`. Además, `connect` define la propiedad `dispatch` por defecto. Esta propiedad nos permite mandar acciones al reducer.

Para asociar atributos del estado con propiedades del componente importamos el HOC `connect`, las acciones y definimos el método `mapStateToProps`:

```
// AL COMIENZO DEL FICHERO...
// Redux
import { connect } from 'react-redux';
import { startSearch, successSearch } from '../../actions/actions';

// Código de la clase...

// AL FINAL DEL FICHERO...

// Esta función nos convierte valores del estado de Redux a props del
// componente
const mapStateToProps = state => {
  // En este caso nos interesan todas las variables del estado, por lo que podríamos
  // devolver una copia de State. Las separamos y las volvemos así a modo
  // ilustrativo
  let { search, loading, results, queried } = state;
  return { search, loading, results, queried };
}
```

Y lo pasamos como parámetro a `connect` al exportar la clase `SearchContainer`:

```
// Connect es un HOC! Modifica los props de nuestro componente para incluir
// dispatch, así como los valores que obtengamos del estado
export default connect(mapStateToProps)(SearchContainer);
```

De esta manera ya tenemos disponibles los atributos del estado como propiedades del componente. Os propongo como ejercicio modificar el componente para hacer uso de estas propiedades y de `dispatch`. Tenéis la solución en src/containers/SearchContainer/SearchContainer.js