Redux-Promise es otro middleware de Redux. Al igual que `redux-thunk`, esta librería nos permite realizar peticiones asíncronas al servidor.

Las acciones que contengan un atributo `payload` de tipo `Promise`, como el valor devuelto por `fetch`, son interceptadas por este middleware. Tras resolver el objeto `Promise`, lanza una copia de la acción con el valor resuelto en el atributo `payload`. Si el `Promise` ha sido rechazado, la acción lanzada incluirá un atributo `error` con el valor `true`.

Primero instalamos la librería:

```
npm install --save redux-promise
```

Vamos a agregar el middleware a Redux. Modificamos el fichero `src/store.js`:

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

// Importamos el middleware
import promiseMiddleware from 'redux-promise';

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

// Creamos el reducer
const store = createStore(
  Reducer,
  // Aplicamos el middleware
  applyMiddleware(promiseMiddleware)
);
export default store;
```

Ahora vamos a crear la acción que lanzará la petición al servidor. Como hemos comentado, necesitamos crear una acción que retorne un objeto que define el atributo `payload`. Este atributo contendrá el objeto `Promise` devuelto por `fetch`. Además, podemos borrar la acción `successSearch` ya que no será necesaria.

```
// Comenzamos una nueva búsqueda
/* export const startSearch = ... */

export const search = value => {
  return {
    type: 'SEARCH',
    payload: fetch(`https://api.github.com/search/repositories?q=${ value }`)
    .then(res => {
      return res.json();
    })
    .then(res => {
      return res.items;
```

```
    })
    .catch(err => {
      // Mostramos el error por consola
      console.log(err);
      return null;
    })
  }
}
```

También tenemos que modificar el reducer, pues la acción de tipo `SEARCH` será ahora la encargada de modificar los resultados en el `state`:

```
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': {
      let results = [];

      if (!action.error) {
        results = action.payload;
      }

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

Por último, solo queda modificar el método `onSubmit` de `SearchContainer.js`:

```
/**
 * Este método actua como callback del evento onSubmit del formulario.
 * Recibe como parámetro el campo que debe de buscar.
 */
onSubmit = value => {
  // Lanzamos la accion!
  this.props.dispatch(startSearch(value));
  // Realizamos la petición a la API
  this.props.dispatch(search(value));
}
```

Tenéis todo el código disponible en la rama redux-promise del proyecto.