María Belén Cervo

Ejercicio Front-End: comentarios

Comandos

Para correr la aplicación se deben usar los siguientes comandos sobre el directorio raíz:

- **npm run install-all**: armé un script en el package.json que instala los módulos necesarios para el back-end y para el front-end y que buildea la aplicación.
- **npm start**: para levantar el server y correr el proyecto en el puerto 5000.

En caso de querer ejecutar la aplicación en modo development, además de arrancar el server hay que entrar al directorio "cliente" y utilizar "npm start". Desde el puerto 3000 se va a correr el server de development y en el puerto 5000 va a correr la api del server.

Estructura del proyecto

- Con el objetivo de estructurar un proyecto escalable, el mismo se encuentra **organizado por features**. Es decir, cada funcionalidad del sistema, representada por un componente de React, está pensada como un módulo que concentra todos los recursos correspondientes al mismo (css, media, acciones, etc).
- También por cuestiones de escalabilidad, elegí manejar las acciones sobre el estado de esos componentes a través de **react-redux**, módulo que permite conectar cualquier componente de la aplicación a un único store de estados (lo cual facilita la comunicación entre ellos y la realización de operaciones sobre esos estados).
- Cada componente cuenta con su propio directorio "**duck**" que concentra sus correspondientes actions, operations y reducers (Redux). Esto previene la manipulación de archivos demasiado extensos que dificulten el desarrollo en una aplicación de gran dimensión.
- Realizo una separación en **componentes de presentación y componentes contenedores**. Esto permite un código más limpio a la vez de favorecer la reutilización de los mismos bajo distintas condiciones. Por ejemplo: el componente Breadcrumb es utilizado tanto en la vista del listado como en el detalle del producto. Aunque no es el caso de este ejercicio, si fuera necesario implementar un lógica de formateo de datos distinta en cada vista, utilizaría un contenedor distinto en cada una pero un mismo componente presentacional para ambas.

Módulos utilizados

- **react-router-dom**: para el ruteo de las diferentes vistas
- **react-redux**: para el manejo de estados de los componentes

- **redux-thunk**: como middleware para manipular las acciones asincrónicas en Redux. A través del uso de operaciones, me posibilita encadenar acciones y realizar los dispatch (o incluso cancelar esas acciones) según una lógica propia.
- **react-redux-form**: por cuestiones de escalabilidad utilizo react-redux-form, que simplifica el manejo de formularios dando acceso a hooks de validación, handlers para el formateo de los campos, handlers y acciones de Redux para la manipulación de eventos, etc.

Server

- Para construir el back-end utilizo el framework Express.js
- Uso el módulo Router, un built-in que permite el empleo de middlewares en las peticiones HTTP y su ruteo al endpoint pertinente.
- Utilizo una organización de archivos que se corresponde con la estructura de las rutas (por ejemplo: "*/api/items/*…"). Esta modularización favorece la escalabilidad, permitiendo el sencillo agregado de endpoints a la api.
- Construyo un wrapper alrededor de la función fetch que posibilita rechazar la promesa en caso de que el status code de la respuesta HTTP no está en el rango de 200-299. Para eso implemento una clase ServerError que además guarda el objeto response.

Otras consideraciones

Usabilidad y experiencia del usuario:

- Si el usuario escribe la búsqueda en la url realiza la búsqueda e incluye el término tal cual lo escribió en el input search.
- Permite la búsqueda de términos con tildes, diéresis, etc. Si el término es escrito en la url, es reflejado de la misma manera en el search box (y la búsqueda continúa enviándose normalizada).
- Si el parámetro "search" de la url no existe o se envía vacío la aplicación es redirigida a la Home.
- El campo de búsqueda se limpia al redirigir tanto del logo del header a la Home como del listado al detalle del producto.
- Los items del breadcrumb permiten la búsqueda de sus categorías sin necesidad de escribirlas en el search box.
- El search box bloquea las búsquedas vacías o repetidas.
- Cada operación posee un loader que indica que la misma se está procesando.
- Cada acción posee su correspondiente mensaje de error (ante errores en las peticiones del cliente, en el servidor o ante la falta de resultados)
- El search box permite la búsqueda de términos con tilde.
- Implemento un custom sort para ordenar los productos de mayor a menor en base al número de ocurrencias de una categoría.

Performance

- Para optimizar las llamadas asincrónicas, utilizo Promise.all para lanzar api requests en paralelo a diferentes endpoints del server de Mercado Libre en lugar de recurrir al encadenamiento de promesas.

SEO

- Dejo implementados los tags de SEO, aunque el servidor no renderiza el contenido de forma estática en las diferentes vistas. Los algoritmos de indexamiento también funcionan con SPA, pero lo más apropiado para el posicionamiento sería realizar un server rendering. A los efectos de este ejercicio decidí no implementarlo (por cuestión de tiempo).

Comentarios adicionales

- No formateo la descripción del producto ya que no posee saltos o tabulaciones que indiquen donde cambian los párrafos.
- Utilizo el middleware redux-logger en modo development como debug para mostrar por consola todas las actions realizadas. En modo de producción no se utiliza dicho módulo.