Documentación Mercado Libre Challenge

Michael Jonathan Schell.

Índice

Instalación del proyecto

Utilizando consola Gitbash ejecutar los siguientes comandos:

1. **git clone https://github.com/ijschell/mercado-libre-challenge.git ./**
2. **cd middleware**
3. **npm install**
   Instalará los components necesarios para que el servidor funcione. El servidor utiliza el puerto 5000 para correr, debería tenerlo desocupado.
4. **cd ../frontend/**
   Viajamos a la carpeta donde se encuentra el frontend.
5. **npm install**
   Instalamos los paquetes necesarios para el frontend, el mismo correrá en el puerto 3000.

Corriendo los servidores de forma local

Vamos a utilizar 2 consolas de gitbash para correr los servidores, una para el front y otra para el servidor middelware.

Frontend:

Estando parados en la raíz del proyecto ejecutar los siguientes comandos:

1. **cd frontend**
2. **npm start**

Middelware:

Estando parados en la raíz del proyecto ejecutar los siguientes comandos:

1. **cd middelware**
2. **node server.js**
   De tener instalado nodemon, recomiendo utilizarlo.

Ahora ya tendríamos corriendo ambos servidores para que puedan estar comunicados entre ellos.

Arquitectura del sistema

A continuación presentaré la estructura de archivos, obviaré los archivos que no utilizaremos como algunos generados por React o los archivos .gitignore del proyecto.

Por otra parte también evitaré escribir repetidas veces el archivo index.js y styles.scss ya que me gusta armar los componentes todos con estos dos archivos haciendo la diferencia de nombres en las carpetas que los contienen, por ejemplo **header > index.js** o **footer > index.js**

1. frontend
    a. src
        i. components
            1. common
            Aquí van los componentes que son utilizados por al menos más de un solo componente
                a. author
                Lo utilizo para imprimir mi nombre con el fin de hacerlo más sencillo de encontrar simplemente
                b. breadcrumb
                Es quien se encarga de armar el breadcrumb en base al array de categorías que lee desde el store.
                c. Header
                Contiene el header y el buscador
                d. Loader
                Un componente que muestra una barra de carga, el mismo puede ser invocado simplemente cambiando su propiedad en el store.
            2. pages
                a. home
                Simplemente una página de bienvenida
                b. product_extended
                Se encarga de obtener los datos del producto que recibe por URL
                c. search_result
                Llama y renderiza los resultados de la query enviada a través del buscador que está en el header.
        ii. Redux
            1. Reducers.js
            Aquí están los handles para los diferentes elementos del store
            2. Store.js
            El estado inicial de la aplicación (store)
        iii. App.js

iv. Common_styles.scss

Una hoja de estilos donde se encuentran las clases que son comunes a cualquier componente

2. Middleware

a. Core

i. Connect_ml.js

Aquí se encuentran las funciones que se encargan de realizar los llamados al Api de Mercado Libre

ii. Preprocess.js

Con los datos obtenidos desde el archive connect_ml rearmo el json que será enviado al frontend, quitando información que no es de interés y filtrando algunas cosas.

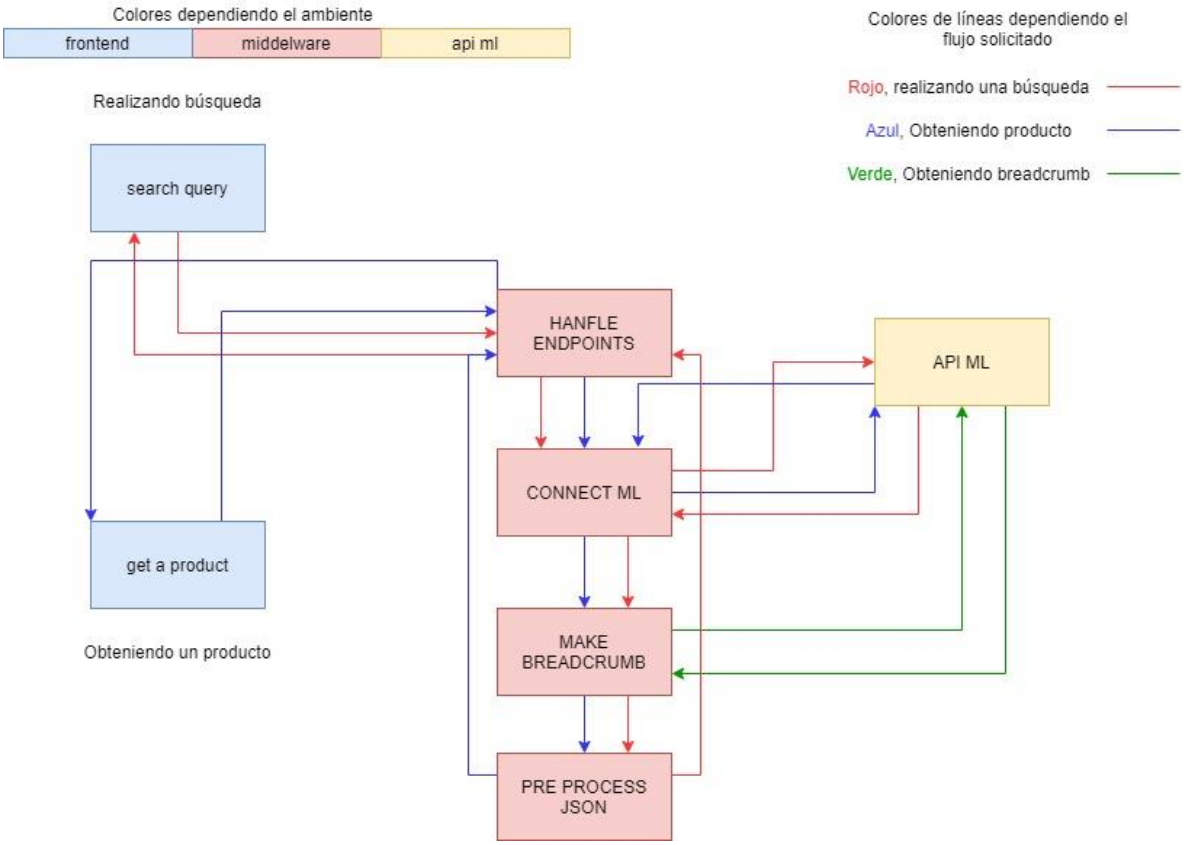
b. Handle_endpoints.js

Este archive se encarga de recibir las peticiones desde el frontend y las orquesta a través del resto del servidor.

c. Server.js

Configuración del servidor, Puerto y middelware.

Diagrama de flujo de datos



Descripción individual de los módulos

En la siguiente sección haré una breve descripción de cada función o método creado, no obstante en el código también encontrará comentarios que lo guiarán.

Al igual que en la estructura de archivos, evitaré escribir los index.js.

1. Frontend
    a. Src
        i. Components
            1. Common
                a. Header
                    i. **Searching**:
                    Se encarga de "setear" el estado del componente con lo que el usuario va typeando en el buscador
                    ii. **go_to_search**:
                    Realiza un "push" en el location history para navegar enviando la query a buscar
                    iii. **handleKeyDown**:
                    Detecta cuando se hace "enter" en el buscador y ejecuta go_to_search
                b. Breadcrumb
                    i. **render_breadcrumb**:
                    Toma las categorías cargadas en el store de la app y los renderiza armando el estilo requerido
            2. pages
                a. search_result
                    i. **get_queryparam_and_set_state**:
                    Obtiene el valor del parámetro query de la URL y "setea" el estado del componente con dicho query, después de eso llama al método get_products_from_api.
                    ii. **get_products_from_api**:
                    Realiza el llamado al api construido en middleware y "setea" el estado del componente con los productos, también hace update del store enviando las categorías para el breadcrumb y autor.
                    iii. **change_to_search_again**:
                    Solamente vuelve a ejecutar get_queryparam_and_set_state porque el query param fue modificado
                    iv. **render_products**:
                    Con los datos "seteados" en el state del componente realiza el render del html.

b. Product_extended

i. **get_queryparam_and_set_state**:
Similar a las funciones del componente search_result, esta función se encarga de leer el ID del producto enviado por URL y se lo envía a get_product_from_api

ii. **get_product_from_api**:
Realiza el llamado al api middleware, con los datos obtenidos "setea" el estado del componente y también envía las categorías y autor al store del app.

iii. **render_product**:
Con los datos en el estado del componente renderiza el html de los productos.

2. Middelware
   a. handle_endpoints
      i. **process_request**:
      Es quién se encarga de recibir los request y orquestar las peticiones a las otras funciones y archivos.
      ii. **send_error**:
      Arma un mensaje de error, es utilizado para mantener siempre el mismo formato y no repetir código.
   b. core
      i. connect_ml
         1. **get_product_by_id**:
         Simplemente recibe el ID del producto a buscar y lo envía a la función go_to_ml con la url "/ítems/:ID"
         2. **get_product_description_by_id**:
         Similar a la function anterior pero intent obtener la descripción de un product, envía "/ítems/:ID/description" a go_to_ml
         3. **search**:
         Recibe la query que está siendo buscada, encodea el string como URL para evitar fallos y lo envía a go_to_ml ("sites/MLA/search?q=query") además pasa un parámetro "limit" para limitar la cantidad de resultados obtenidos desde el Api de ML, de esa forma la consulta es más performante.
         4. **make_breadcrumb**:
         Le envía un array de IDs de categorías y se las envía a category_most_frequent para obtener solamente un ID, el más frecuente, ese ID es enviado a go_to_ml para obtener la información de la misma.
         5. **go_to_ml**:
         Devuelve una promesa, es la función que se encarga de llamar al Api de ML.
         6. **category_most_frequent**:
         En base a un array proporcionado, obtiene el elemento más frecuente y lo devuelve.
      ii. preprocess

1. **search**:
   Obtiene un array de los productos resultantes de la búsqueda y filtra cada uno de ellos para enviar al frontend solo los datos que interesan.
2. **product**:
   Al igual que la función anterior, pero con un solo objeto.

Comentarios

Para finalizar la documentación dejaré algunos comentarios que fui encontrando mientras realizaba la prueba.

Con respecto al render de los resultados, entiendo que en el mockup se ve la ciudad de donde se vende el producto, pero a su vez en el json que se debe armar para devolver al frontend, no se solicita obtener este dato, por lo tanto, yo lo obtuve y lo imprimí para que sea similar al mockup.

Por otra parte en la página del producto extendido, en el mockup aparece el breadcrumb pero este dato tampoco es solicitado al momento de armar el json, por lo que modifiqué un poco mis funciones del middelware para obtenerlo y de esa manera queda fiel al mockup.