Práctica 2 - React.js

Introducción y Objetivos

Requisitos:

- Nociones básicas de las tecnologías explicadas en el curso(HTML, CSS, JS, **React**)
- Editor de código Visual Studio Code (o similar)
- Tener instalado node.js versión 16 (LTS) y git

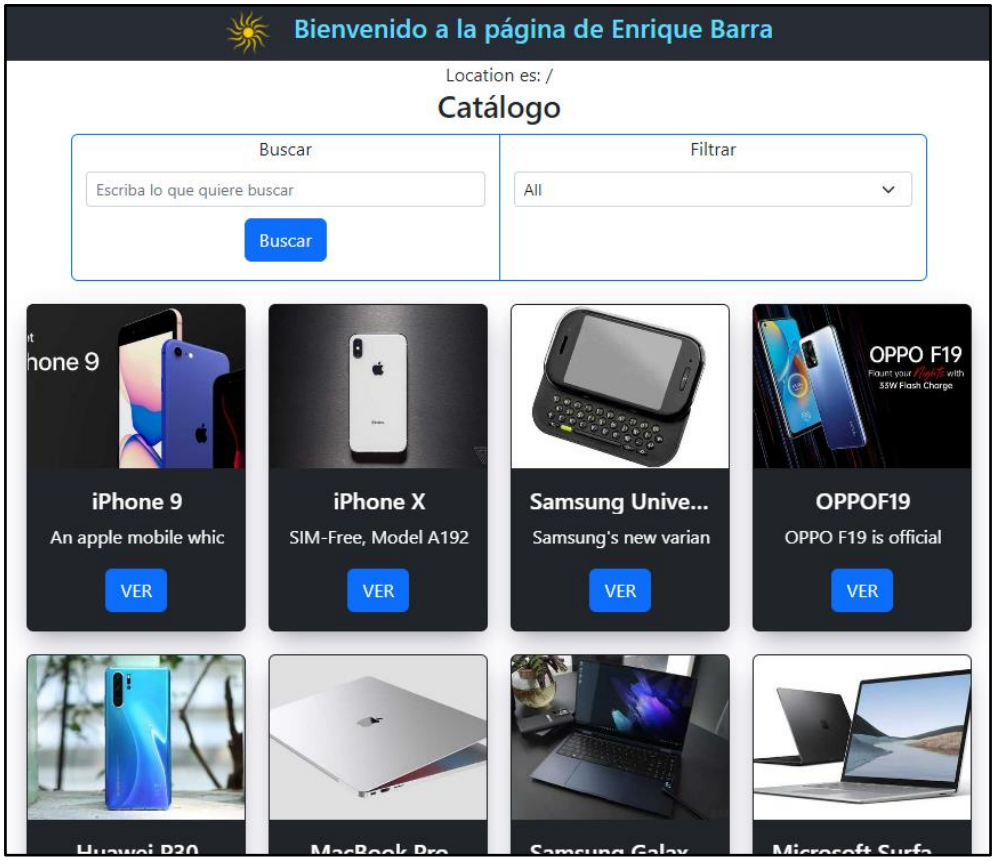
Motivación:

Intenta mirar esta práctica como el modo de afianzar los conocimientos en React e introducir algunos conceptos nuevos (rutas, hooks, loading spinner, bootstrap, …). En este caso también se acabará haciendo una aplicación real y completamente funcional. De nuevo piensa en este enunciado como en una definición de una captura de requisitos resultados de varias reuniones con un cliente que es “el que paga”. Nos han pedido hacer esta aplicación con estas características mínimas y estos requisitos. Y para que nos paguen (y por lo tanto obtener el 10/10) debe cumplir con estas especificaciones y para ello nos pasan una batería de tests que debe pasar.

Objetivo:

El objetivo de esta práctica es realizar una Single Page Application (SPA) de un catálogo de productos. Esta aplicación al cargar realiza una query a un servidor (API) y mostrará los resultados. También presenta un campo con el que buscar en el catálogo y un selector para filtrar los resultados por categoría.

Los productos se presentan en tarjetas como muestra la siguiente captura y si se hace click sobre un producto se accede a la página del producto para visualizar su información.



Página principal

Página de un producto

Primeros pasos: Preparación del entorno de desarrollo

Para poder hacer esta aplicación se proporciona un esqueleto o scaffold en github con el resultado de ejecutar el comando "npx create-react-app P2_productos" y añadir la herramienta de autocorrección y los tests a pasar para superar esta práctica.

1. Clona el esqueleto de la aplicación a tu ordenador: `git clone https://github.com/REACT-UPM/P2_productos_MOOC.git`
2. Entra en el directorio creado: `cd P2_productos_MOOC`
3. Instala los paquetes necesarios que indica el package.json: `npm install`
4. Inicia el servidor de desarrollo: `npm start`

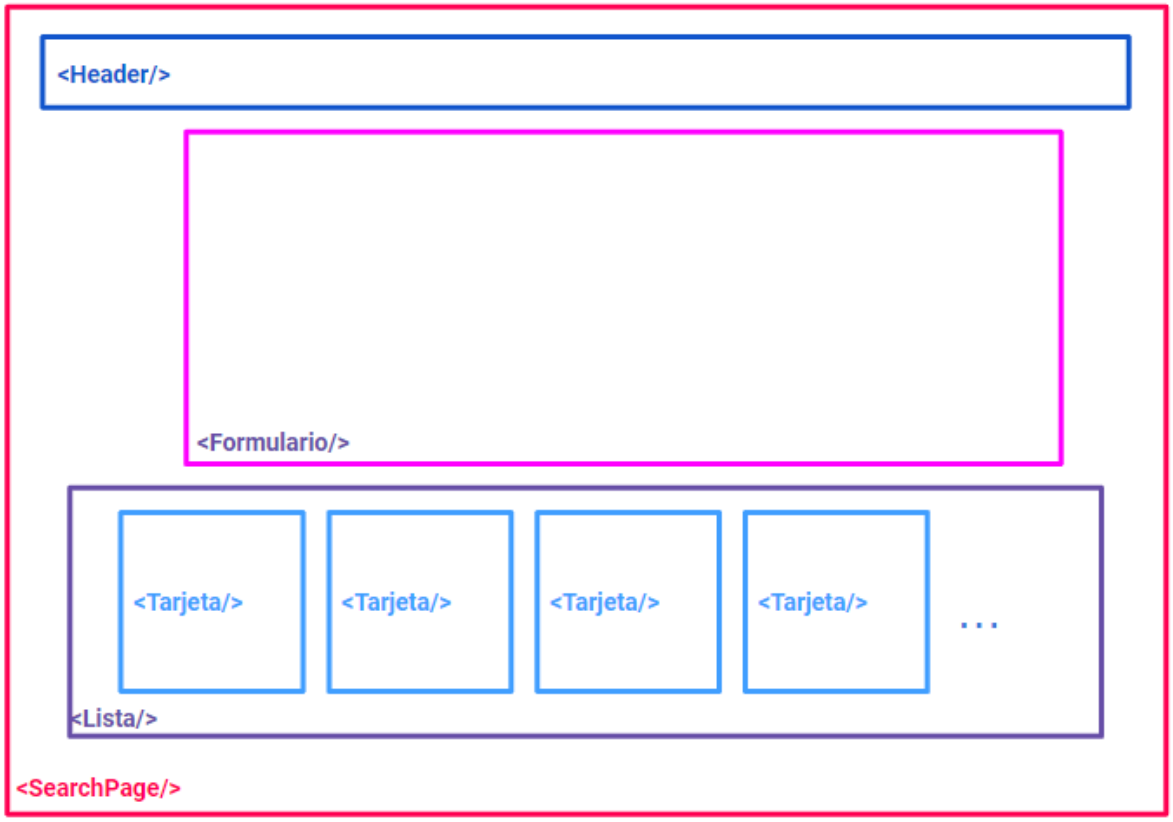
A partir de ahora, si abrimos el navegador en la URL http://localhost:3000, podemos visualizar la aplicación en tiempo real mientras desarrollamos. Cuando queramos parar el desarrollo basta con hacer `Control + C` en el terminal.

Ahora hay que abrir la carpeta P2_productos en el editor Visual Studio Code y empezar a desarrollar nuestra aplicación.

Comenzando a desarrollar

Antes de empezar a programar, uno debe pensar cómo estructurar la aplicación, y decidir *grosso modo* qué componentes va a necesitar y la jerarquía de los mismos. En este caso tenemos rutas para la página principal ("/") y para los productos ("/products/:productId") y por lo tanto el componente App.js lo que hará será renderizar una página u otra dependiendo de la ruta solicitada.

Un posible diseño para el componente SearchPage es el siguiente:



En esta práctica los componentes App, Header y SearchPage se tienen que llamar así por definición.
También se recomienda definir el estado de nuestra aplicación (aunque sea una primera aproximación que luego irá mutando según añadamos funcionalidades). ¿Qué **información** necesito para pintar esta aplicación?

API de productos

En esta aplicación utilizaremos un API disponible en DummyJson. En este caso no es un api real sino un API Rest para practicar, pero está bien documentada y es más o menos potente.
La documentación del API está en:
https://dummyjson.com/docs/products
El endpoint es:
https://dummyjson.com/products

Como en la P1 a la hora de desarrollar aplicaciones que “atacan” APIs hay que utilizar unos datos fake para ir haciendo pruebas con datos controlados y luego cuando tengamos todo desarrollado implementamos la llamada al API y refinamos lo que falte. Para este propósito se provee en el proyecto de github en la carpeta *src/constants* un fichero js que exporta un mock de los datos. Examine dicho fichero para ver el formato de los datos.

Configuración de la aplicación (fichero config.js)

Para esta aplicación se provee un fichero en el proyecto de github en la carpeta *src/config* llamado *config.js* que exporta una configuración inicial propuesta. Examinar dicho fichero para ver las variables que exporta.

Requisitos que debe cumplir nuestra aplicación

La práctica se valorará sobre un máximo de 10 puntos. La mayoría de los requisitos son de funcionalidad, para la interfaz de usuario puede maquetar como desee. En este caso se recomienda usar Bootstrap para maquetar. https://react-bootstrap.github.io/getting-started/introduction

Requisitos:

- **1 punto**: La aplicación tiene un componente Header con el logo y el mensaje de bienvenida con tu nombre (el nombre debe coincidir con el que le hemos dado al autocorrector la primera vez que lo ejecutamos).
    - El componente Header debe renderizar un elemento div con el id “cabecera”
    - Dentro de dicho div debe tener una imagen con la clase “logo” y un h3 con la clase “mensaje”
    - El h3 debe contener el texto que desee pero al menos debe poner tu nombre. Por ejemplo como se ve en la captura inicial de esta práctica “Bienvenido a la página de Enrique Barra”
- **1 punto**: La aplicación, mientras carga, muestra una imagen de un spinner con un id “loading” y una clase “spinner”
    - Esto se puede hacer directamente con una etiqueta img que muestre un gif o similar o con un componente Spinner que nos da bootstrap ya maquetado y animado (https://react-bootstrap.github.io/components/spinners/)
- **1 punto**: La aplicación tiene un componente SearchPage, con al menos un h2, un input y un button
    - el h2 debe tener el id “catálogo” y contenido al menos la palabra “catálogo”, el input debe tener el id “filtro” y el button debe tener el id “buscador”
- **1 punto**: La aplicación tiene un componente SearchPage que renderiza los productos que recibe
    - El componente searchPage tiene que recibir una prop “theproducts” con un array con los productos (inicialmente mockdata.products y posteriormente vendrá de una query al servidor)
    - Se puede hacer un componente hijo Lista al que pasemos los productos y que sea éste el que los pinte o que sea SearchPage la que lo pinte directamente
    - En cualquier caso se recomienda recorrer el array con un map y cada producto renderizarlo o bien con y usando css3 para maquetar o bien con las Cards de Bootstrap (https://react-bootstrap.github.io/components/cards/)
    - IMPORTANTE: Sea cual sea la renderización, por fuera de la lista poner un div con un id “productosresultados” y cada producto debe tener la clase

"unproducto". Esto será lo que use el corrector para buscar los resultados y contarlos y comprobar las funcionalidades.

- **1 punto**: La aplicación maneja el valor del input y filtra los resultados por su título al pulsar el button
    - Llamar a una función que coge el array de productos y utiliza el método "filter" de array para dejar en el array solo los que contienen el texto que ha introducido el usuario (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)
- **1 punto**: La aplicación tiene un selector de categorías de productos y está relleno correctamente
    - La etiqueta debe ser un select y tener las options adecuadas dentro (https://www.w3schools.com/tags/tag_select.asp). Como primera <option>, la seleccionada por defecto debe contener la palabra "All". Luego el resto de options que tengan la palabra de la categoría y el value también el nombre de la categoría.
    - También se puede implementar con https://react-bootstrap.github.io/forms/select/
    - En cualquier caso la etiqueta select debe tener el id "selector"
    - IMPORTANTE: no se pueden poner las categorías "a pincho". Este selector hay que generarlo dinámicamente a partir de los datos de productos. Para ello se puede usar el método "reduce" de array https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce (aunque hay otras maneras por supuesto)
- **1 punto**: La aplicación tiene un selector de categorías de productos y filtra al seleccionar una categoría
- **1 punto**: La aplicación tiene un componente para la página de un producto
    - Esta página debe tener al menos un elemento con id "titulo" que contenga el título del producto y un botón con id "volver" para poder volver a la página principal una vez visualizado el producto
    - Esta página se sirve cuando el usuario pide la ruta "/products/:productId" donde :productId es la posición que ocupa el producto en el array (atención no es el campo id sino la posición). Es decir por ejemplo /products/0 mostrará el primer producto
    - para que este test pase la página del producto debe renderizar en algún lugar el componente Location provisto en la carpeta src del proyecto de github así: <Location/>
- **1 punto**: La aplicación tiene una ruta para NoMatch
    - En caso de que la ruta no sea ni "/" ni "/products/:productId" la aplicación debe renderizar una página de error con un botón volver con id "volver" y un campo con id "info" que contenga "Ruta no encontrada".
- **1 punto**: La aplicación hace fetch de datos del servidor remoto
    - cuando el parámetro use_server vale true en el fichero de config se hace la petición al servidor y se utilizan los datos que devuelve
    - la url del servidor se obtiene también del fichero de configuración del parámetro server_url

Pruebas manuales y capturas de pantalla

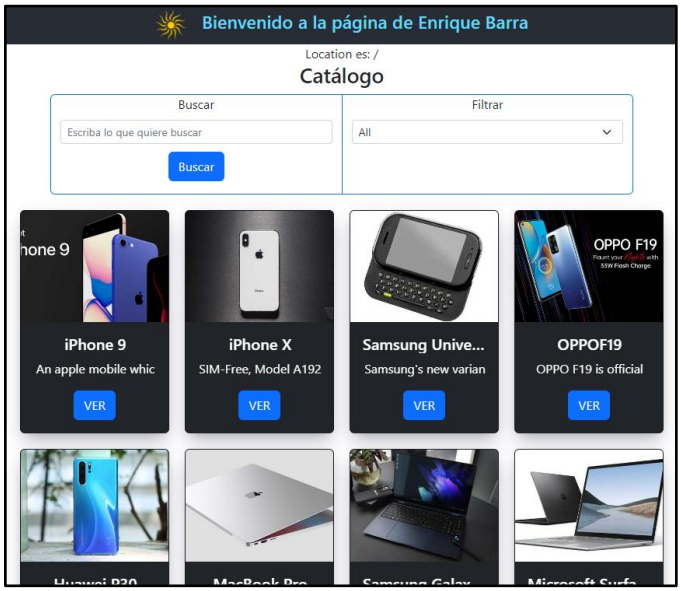
Como se ha comentado se provee una batería de tests que la aplicación debe pasar y que serán los que produzcan la nota final. Pero es importante desarrollar la aplicación viendo poco a poco el resultado y visualizando cada cambio introducido hasta que funcione. Considerando los test provistos por el autocorector como una serie de requisitos “estrictos” o “estáticos” como por ejemplo que tenga determinado id o clase alguna etiqueta o que el componente se llame de determinada manera.

No se recomienda por lo tanto ir desarrollando sin visualizar lo que hacemos y pasar el autocorector a cada paso “a ver si se consiguen los puntos”, porque los errores que dan las baterías de tests son más crípticos que lo que veremos en el navegador o en la consola por nosotros mismos al visitar http://localhost:3000

Adicionalmente a pasar la batería de tests y obtener un 10/10 hay que hacer dos capturas de pantalla con la aplicación completa, es decir en la versión final antes de entregar. Dichas capturas se tienen que colocar en formato png, jpg o pdf en el directorio “miscapturas”. Y el autocorector las subirá junto con el código de la práctica y el resto de evidencias a Moodle.

Estas capturas son obligatorias y deben ser personales, en ellas se debe ver que la cabecera de la práctica pone el nombre del alumno y que el estilo es el entregado en el código. Estas capturas serán revisadas por el profesor para comprobar que se ha realizado la funcionalidad correctamente.

En esta práctica hay que subir dos capturas similares a las siguientes, es decir una de la página principal y otra de la página de un producto:



Pruebas con el autocorector

El autocorector es la herramienta que permite pasar la batería de tests a la práctica y producir una nota. También subirla a Moodle junto con el código desarrollado, las capturas y otras evidencias de evaluación.
Ejecute el autocorector tantas veces como desee en la práctica y suba la nota a Moodle también tantas veces como desee hasta que se cierre la entrega.

Dudas y tutorías

Para dudas sobre el enunciado y sobre la práctica en general vamos a utilizar **el foro del curso**. Lo único que está prohibido es subir vuestra solución completa al foro en un zip o enlace o similar. Si se pueden compartir trozos de código para pedir ayuda o para ilustrar lo que os está ocurriendo.
Como insistimos en la guía del curso y comentamos el día de la presentación **la copia de la práctica es un suspenso automático de todo el curso** tanto para el que copia como para el que se deja copiar. Así que por favor se prudente y no compartas tu código como un zip con tus compañeros, una vez que sale de tu ordenador no sabes quién puede entregar tu código en su nombre.