DESARROLLO DE UN FRONT-END UTILIZANDO REACT

Juan Carlos Muñoz Castañeda

Junio 2023.

Fundación Universitaria Internacional de la Rioja

Especialización en Ingeniería de Software

Desarrollo de Aplicaciones WEB

**Tabla de Contenidos**

[INTRODUCCION 3](#_Toc138671768)

[COMPONENTES DE REACT 5](#_Toc138671769)

[HOOK UTILIZADOS 7](#_Toc138671770)

[CUSTOM HOOK 8](#_Toc138671771)

[VISTAS 9](#_Toc138671772)

[CONCLUSION 11](#_Toc138671773)

[BIBLIOGRAFÍA 12](#_Toc138671774)

# INTRODUCCION

Con el auge de un lenguaje de programación como JavaScript se inicia una etapa en donde las páginas WEB adquieren mayor dinamismo.

El realizar un despliegue de una página requiere la combinación lenguaje de etiquetado como HTML, un manejador de estilos como CSS y la lógica y dinámica de la página manejada con un lenguaje de programación como JavaScript.

Un fichero con extensión HTML es leído e interpretado por el navegador y dicho archivo permite la escritura de HTML, CSS y JavaScript en el mismo fichero.

Para un mejor mantenimiento del código se comenzó a adoptar ficheros por separado, el etiquetado (ficheros HTML) encargados de la muestra de elementos en el navegador, Ficheros CSS en donde está la estructura de diseño de cada elemento o etiqueta del HTML y la lógica y dinámica en Ficheros JS.

Para hacer mantenible el código de las aplicaciones WEB , comienzan a llegar los Frameworks de JavaScript tales como Angular, Vue y React, en donde vuelven a juntar al menos el código HTML junto con código JS en un solo modulo conocidos como componentes, los cuales por su estructura pueden ser reutilizables conforme al diseño, los cuales aunque usando JavaScript o (TypeScript exclusivo para Angular), también poseen objetos, métodos y propiedades que no solo hacen interactiva la página sino también mas dinámico el desarrollo de las misma.

La presente Aplicación para esta Asignatura fue realizada en React en su versión 16 y desplegada para su entorno de desarrollo en Vite 4 (similar Create-React-App).

Se utilizo la API de <http://openlibrary.org/> en donde se pueden encontrar libros para lectura online. La API solo se usó para listar primero 10 libros encontrados al ingresar una palabra clave o parámetro a manera de ejercicio, su portada en caso de encontrarse, mostrar el autor, año de publicación e ISBN.

Adicionalmente como Ingeniero Catastral y Geodesta decidí colocar una simulación de hipotéticas librerías en donde se encontraría la ubicación del libro, las mismas mostradas en una salida cartográfica cuyo mapa o mapas base es propiedad y distribución libre por parte de [https://openlayers.org](https://openlayers.org/) , desplegadas con ayuda de la librería “Leaflet” [https://leafletjs.com](https://leafletjs.com/) , que nos permite crear mapas geográficos dinámicos en los cuales podemos navegar y desplazarnos, cuya profundización de utilidad la desarrollo más adelante en el capítulo de componentes.

A manera de valor agregado, el ejercicio se desplego en un hosting compartido en el cual se puede acceder a <http://bookstore.geovisor.online> para ver la aplicación en producción y también al repositorio en donde lo aloje en GitHub <https://github.com/juancmu/bookFinder.git>

# COMPONENTES DE REACT

1. APILIBRARY: Este componente es el componente principal de la lógica, se encarga de recibir el parámetro de búsqueda desde el componente FINDER en el cual se encuentra el input, realizando el llamad0 a la API recibiendo el JSON, retornando los mensajes de coincidencia o no conforme al parámetro ingresado y enviando la información recibida por la API hacia el componente hijo BOOKLIST.
2. Si bien este componente es el principal el cual hace la primera conexión con la API, hay otros componentes que veremos más adelante, que también realizan peticiones a la API.
3. MAIN: encargado del manejo de las rutas mediante REACT-ROUTER así como el envió de la renderización al div “root” del index de despliegue.
4. NAVBAR: Como su nombre lo índice, maneja el panel de navegación con 4 vista, la cual la primera es el index, INICIO que retorna al listado de libros de búsqueda inicial, INFO en donde coloco los datos de información genéricos y REPORTE en donde devuelve el PDF de este informe.
5. FINDER: es el componente compuesto por un Input que de encarga de recoger el parámetro de búsqueda y enviarlo a APILIBRARY para que realice la búsqueda a la API.
6. BOOKLIST: este componente recibe la lista de libros encontrados, y también realiza la búsqueda de la portada de cada libro dentro de la mismo API, es decir, este componente también realiza una búsqueda a la API en el link provisto por OpenLibrary, el cual real1iza esta búsqueda por ID, no todos los libros tienen portada almacenada, por el caso no encontrarla , devuelve una portada de no encontrada por defecto, luego de tener esta información, realiza el llamado al componente BOOK que es el encargado de mostrar la información encontrada por cada libro.
7. BOOK: este componente es el que despliega la portada, Titulo, Autor, Año de publicación e ISBN de cada libro que recibe y es reutilizado por BOOKLIST para la vista general de todos los libros encontrados.
8. BOOKID: este componente es llamado cuando el usuario hace click en el libro deseado recibiendo el id del libro y realizando la búsqueda a la API para retornar de nuevo la información y la portada, mostrando como resultado la portada, el titulo y la descripción, que en caso de no encontrarla también retorna valores por defecto.

Adicionalmente en este componente realiza el llamado del componente MAPS que se encarga de mostrar un mapa geográfico que describo en el siguiente componente.

1. MAPS: es llamado por el componente BOOKID, y despliega solo a manera de ejercicio la ubicación de las posibles tiendas donde se puede encontrar el libro disponible. Esta información geográfica desplegada gracias a la librería LEAFLET, es aleatoria y no esta vinculada con la información de la API, solo se usa para darle un valor de integración al ejercicio de aplicación y para usar a amas detalle temas del Hook UseState.
2. LOADER: este componente muestra un objeto Canvas mientras la aplicación realiza la búsqueda a la API y retorna los resultados a los componentes destino.
3. REPORT: si bien fue realizado con un componente, este solo es invocado cuando en el NAVBAR se hace click en el REPORTE y carga el pdf en otra pagina de este informe anexo.

# HOOK UTILIZADOS

1. USEEFFECT: este hook se usa cuando se realiza una consulta a la API esperando de manera asíncrona la información y retornando el componente requerido, montando el componente, actualizándolo y desmontando cuando sea requerido.
2. USESTATE: usado para cambiar el estado de algunas variables y objetos dentro de los componentes de la aplicación, como ejemplo se uso este en el componente MAPS, para cambiar el estado de las coordenadas en donde esta ubicada la tienda, cambiando la posición del arreglo en donde están estas coordenadas y el nombre de la tienda.

Igualmente cambia el parámetro por defecto de búsqueda cuyo valor inicial es “programación” y el hook es activado cuando se realiza la búsqueda desde el componente FINDER enviando el parámetro a APILIBRARY.

1. USECONTEXT: usado en algunos componentes para la comunicación entre los mismos, sin importar la jerarquía entre ellos.
2. USECALLBACK: es usado por el componente APILIBRARY, se usa junto a la búsqueda asíncrona a la API, con el fin de reducir el uso en memoria y guarda la información encontrada en contexto local para mejor optimización y así cada vez se renderice un componente o vista no vuelva a realizar la búsqueda a la API, sino que use la información almacenada siempre y cuando no cambie el parámetro de búsqueda.

(La implementación de los hooks fue implementado con gran ayuda del siguiente creador de contenido (Cabrera, 2021) )

# CUSTOM HOOK

1. USECUSTOMHOOK: Se realizo un custom hook usando useState de react, para almacenar el parámetro de búsqueda en el localStorage del navegador, con el fin que me guarde el parámetro en memoria y lo mantenga independientemente de refrescar el navegador, una vez se refresque el navegador, el Custom Hook solicita al LocalStorage el parámetro y lo envía al input del formulario del componente FINDER.

# VISTAS

En la aplicación solo se manejaron tres vistas independientes, ya que como se mencionó anteriormente, algunos componentes recibían datos de otros componentes, como se ve en la estructura del proyecto, maneje los archivos de estilo en CSS puro, independientes casi por cada componente o vista creada. Igualmente, se usaron para algunos elementos de algunos componentes, como fuentes o estilo de botones, CSS puro en combinación con la librería para estilo CSS Bootstrap. Las vistas son las siguientes:

1. HOME: Componente llama a los componentes principales NAVBAR, FINDER Y OUTLET, el cual este último es el componente nativo de REACT-ROUTER en donde se renderizan los componentes hijos una vez se realice la consulta a API.

Figura 1. Vista del HOME de la aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. ABOUT: cuando en el NAVBAR se hace click en INFO se muestra la información básica de las plataformas de desarrollo usadas en la aplicación e información general, el cual es renderizado en la “sección” OUTLET propia de REACT-ROUTER.

Figura 2. Vista del ABOUT de la aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. NO FOUND: Vista por defecto cuando una ruta es mal ingresada al navegador, dejando la posibilidad de devolver a la página de inicio. Esta plantilla se encontró en internet y de adapto a la aplicación (Krivec, 2021)

# CONCLUSION

Una vez realizado el ejercicio del uso de un framework como React para el desarrollo de aplicación WEB desde el lado del cliente (Front-end), se puede concluir al menos a de manera subjetiva, un cambio en el paradigma de programación.

Acostumbrados al uso de separación entre el diseño y la lógica que se suele usar desde el JavaScript puro o Vanilla JS termino acuñado por Eric Wastl referente al uso del JavaScript sin librerías ni frameworks (Peris, 2020) , el uso de un framework genera la reutilizabilidad del código al separar los diferentes elementos de una aplicación en componentes, pero hace que el programador entienda la lógica de funcionabilidad el framework que aunque usa JavaScript, el uso de las funciones y objetos preestablecidos hace que se comporte como un programa diferente. Quizás lo más costoso de implementar es que cada objeto, sea imagen, pdf etc.… debe ser importado al respectivo componente que lo desea usar recordando lenguajes de alto tipado como JAVA o C++.

El desarrollo de esta aplicación de consumo de una API publica desplegada en un ambiente de React en donde solo se interactúa desde el lado del cliente, en si no fue un ejercicio difícil. Lo más complicado fue el tema de la relación entre componentes, paso de variables, funciones y objeto entre ellas, entender como funcionan sus Hooks nativos que prácticamente son los que dan la funcionabilidad a la aplicación y el manejo de estados en donde fue un dolor de cabeza entender para aquellos sin la experiencia o formación en este concepto.

# 

# BIBLIOGRAFÍA

Cabrera, L. (2021). *https://www.youtube.com/@LuisCabrera*. Obtenido de https://www.youtube.com/@LuisCabrera

Krivec, R. (26 de Marzo de 2021). *colorlib*. Obtenido de colorlib: https://colorlib.com/wp/template/colorlib-error-404-12/

Peris, L. (11 de noviembre de 2020). *LUIS PERIS*. Obtenido de https://luisperis.com/que-es/vanillajs/