Módulo 5 – Life Cycle y Hooks

# Ciclo de vida

Este es un concepto ampliamente considerado en diferentes paradigmas y lenguajes de programación. Sobre todo en entornos donde una entidad o instancia contiene estados, los cuales pueden cambiar. Así mismo, temas como la gestión de recursos y su respectiva liberación son cruciales durante su ciclo de vida.

El entorno web no es la excepción, como vimos previamente con los componentes Clases de ReactJS, estos heredaban funciones que ayudaban a la administración del ciclo de vida y alterar su comportamiento antes y después de cada render.

Con los componentes funciones se puede realizar lo mismo. Esto lo logramos con la ayuda de dos principales Hooks “useEffect” y “useLayoutEffect”. Los Hooks de “useMemo” y “useCallback” no están directamente ligados al ciclo de vida del componente, pero si pueden provocar cambios en el ciclo de vida del componente al que esté ligado.

Describir el gráfico del ciclo de vida

# Analizando “useEffect”y “useLayoutEffect”

Previamente mencionamos brevemente que estos dos Hooks tiene un comportamiento muy similar, ambos están involucrados con el render de un componente. Con la diferencia de que “useEffect” tiene su participación después de que el componente haya sufrido algún render y asincronno. Por otro lado, “useLayoutEffect” sucede antes de algún render y en ASINCRONA.

Los usos donde estos Hooks pueden ser útiles son muy variados. Un buen ejemplo es en formularios.

Casos de uso para “useEffect”:

* Edición de campos precargados, para esta situación pueda que recibamos un ID como param en la URL. Por lo que debemos realizar un fetch a la data del ID. Entonces deseamos que, inmediatamente después de que tengamos la data, llenar el formulario con los datos.
* Invocar un servicio o lógica después de que un estado cambie.
* Liberar recursos o ejecutar funciones de “Clean up” antes de que el componente sea desmontado.

Casos de uso para “useLayout”:

* Manipular algún elemento del DOM antes de que se amostrado en pantalla.
* Evitar parpadeos o cambios repentinos de un componente. Pueda que cuando un estado llegue a cierto valor deseamos mostrar otro diferente. Para esto podemos usar useLayout, para “interceptar” el cambio y cambiarlo a lo que deseamos antes de que este llegue a renderizarse.

# Uso de ciclo de vida

Implementaremos nuevos comportamientos en la sección del formulario como:

* useLayout para pre-cargar los datos del user en modo edición.
* useEffect, Hacer focus a un componente cuando el componente se muestra.
* useEffect, Limpiar state del form cuando se sale del form (esto será mas útil si tenemos form en modals, normalmente los modals quedan no visibles pero sus states cargados, por lo que es importante reiniciarlos)
* useMemo, para hacer que un componente custom cambie solo cuando verified cambia.

# Memo components

Memoization es un proceso en el que se cachea o “memoriza” un componente. Esto se lo hace principalmente para mejorar el rendimiento de las pagina web. Es importante aislar los componentes y estos se rendericen solo cuando valores asociados a ellos cambien.

Existen dos maneras para llevarlo a cabo:

* React.memo, el cual es un HOC y hara que el componente se renderice solo si alguno de sus props cambian.
* React.useMemo, funciona similar a React.memo pero con la diferencia de que este al ser un hook tiene una lista de dependecnian como useEffect, y si alguno de los valores de esta lista cambia el componente de renderiza nuevamnete.

# ¿Cómo crear Hooks?

Slide 2

Como muchas veces en programación, para crear algo se parte de bloque fundamentales, entonces para crear un Hook, ¡lo hacemos a través de otros Hooks! La idea o motivación de crear nuestros Hooks es generalmente para encapsular código que debe mantener un estado a través de los renders.

Por ejemplo, podemos necesitar un de Hook de paginación de datos. Con este hooks podemos enviarle todo una Array de datos, un número N que es la cantidad de items por página. Y este nos devuelve una página con N items, una función para ir a la página siguiente, atrás, principio y final.

Input debounced, a veces uno tiene un buscador, pero no desea realizar la búsqueda y fetch en cada carácter que cambie. Entonces podemos crear un hook que actualice el valor final solo cuando haya pasado cierto tiempo luego del último cambio realizado. Otro muy usado es un Hook para administrar listas, con este recibimos funciones para agregar elementos, en ciertos índices, eliminar o limpiar el state.

Otro ejemplo que he usado es “useParams” lo uso principalmente para poder invalidar fetch realizados con una librería llamada react-query. Esta librería recibe un Key y función para hacer el fetch. Muchas veces algun cambio de un fetch debe hacer que otro actualice los datos (como los forms). Entonces con este hooks yo administro el key y si este cambia marco invalidado el hook.

Mostrar ejemplos en mis repos y de librería Mantine para mas ejemplos de como usarlos

# Convenciones para Hooks

Slide 3 - 4

Las principales convenciones es que el nombre del hook debe iniciar con el prefijo “use” El resto como la firma de la función puede ser a conveniencia del usuario. Ya sea parámetros posicionales o un objecto. El return generalmente es un arreglo y en Typescript debe ser declarado como “as const” para no tener problemas en los tipos. También se puede retornar un objecto para destructuring por nombre y no por posición como hemos visto.

# Creando Hook de paginación

1. Crear la carpeta “hooks”
2. Crear el hook de paginación con parámetro en JSON, y devuelve el arreglo con items por página, función para ir adelante, atrás, total de páginas.
3. Crear el type de los parámetros.
4. Crear las funciones necesarias como “function” y explicar por qué, por convencion y mejor visual al leer.

# Práctica

Exclusiva by Camilo

Se debe crear un nuevo Hook “useDebounce” el cual recibe como parámetros valor inicial de tipo parametrizado y un valor numérico. Como return devolverá una función para actualizar el valor (como el useState) y el valor almacenado dentro del Hook, pero actualizado solo después de que haya pasado la cantidad de milisegundos especificada en el parámetro numérico.

Es decir, para actualizar el valor dentro del Hook se usará su función retornada pero el valor del state retornado por el Hook solo se actualizará una vez que haya transcurrido la cantidad de milisegundos del parámetro después de la última actualización. Cada vez que se actualiza el valor el contador debe reiniciarse y la actualización solo se efectuará cuando el tiempo necesario se haya cumplido.

Este nuevo Hook sebe ser implementado en la página “/users”. Con el objetivo de agregar un input para buscar un usuario por nombre. Cada vez que el usuario escribe se debe actualiza el valor de Hook, pero la búsqueda vista solo se actualiza cuando el Hook haya retornado el valor actualizado.

Puntos extra: lograr retornar un valor adicional “debouncing” que indica cuando el Hook está esperando que se cumpla el tiempo necesario para actualizar el valor, un análogo de un estado de “cargando”.