Módulo 4 – 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.
* Hacer focus a un componente cuando el componente se muestra.
* Limpiar state del form (esto será mas útil si tenemos form en modals, explicar)

# ¿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 taves de otros Hooks! La idea o motivación de crear nuestors Hooks es generalmente para encapsuar código que debe mantener un estado a traves 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 busqueda 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 ultimo cambio realizado. Otro muy usado es un Hook para adminitrar listas, con este recibimos funciones para agregar elementos, en ciertos indices, eliminar o limpiar el state.

Otros 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 cone ste hooks yo adminitro 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 conversiones 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 pocionales o un objecto. El return generalmente es un arreglo y en Typescript debe ser delcarado 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 parametro 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.

*Cotinuar con teoria y practica de life cycle components*

*La práctica/tarea será hacer un useDebouce que recibe un valor string inicial y un número en milisegundos y devuelve como valor otro string debounced. Usar este hooks para agregar un buscardor en la vista “/users/view” para buscar usuarios según su nombre.*