HOOKS

En React, los Hooks representan una evolución en la forma de construir componentes funcionales, permitiendo acceder a funcionalidades avanzadas como el manejo de estado, efectos secundarios, contexto global y persistencia de referencias, sin recurrir a clases ni estructuras verbosas. Esta semana se orienta al estudio conceptual de los Hooks principales (useState, useEffect, useContext, useRef, useReducer, useCallback, useMemo), así como el diseño de Hooks personalizados. Su comprensión es esencial para lograr aplicaciones eficientes, modulares y legibles.

Desarrollo Conceptual

I. Naturaleza de los Hooks

Los Hooks de React son funciones que permiten "engancharse" al ciclo de vida del componente, acceder a su estado interno, o conectar elementos con lógica funcional. Su creación surge de la necesidad de:

- Simplificar la escritura de componentes usando funciones puras.
- Evitar duplicación de lógica en estructuras complejas.
- Facilitar la reutilización y testeo de funcionalidades específicas.

II. Hooks Fundamentales

1. useState
    - Permite crear y actualizar variables de estado dentro de componentes.
    - Devuelve un array con el valor actual y una función para modificarlo.
    - Se usa para datos reactivos: entradas del usuario, switches, contadores.
2. useEffect
    - Administra efectos secundarios como llamadas a APIs, eventos, actualizaciones al DOM o almacenamiento local.
    - Se ejecuta después de cada renderizado y puede controlarse mediante un array de dependencias.
    - Útil para sincronizar componentes con fuentes externas de datos.
3. useContext
    - Permite acceder a valores definidos en un Contexto de forma directa.

    - Evita el paso de props entre múltiples niveles jerárquicos.
    - Es ideal para temas visuales, autenticación, configuración de idioma o preferencias.
4. useRef
    - Crea una referencia persistente que no cambia entre renderizados.
    - Se usa para acceder a elementos del DOM o guardar valores temporales.
    - No provoca re-renderizado al cambiar su valor.
5. useReducer
    - Alternativa a useState cuando se maneja estado complejo y acciones múltiples.
    - Usa una función reductora para actualizar el estado de forma declarativa.
    - Ideal para formularios, listas con acciones y cambios sincronizados.
6. useCallback
    - Memoriza funciones para evitar su recreación en cada renderizado.
    - Mejora el rendimiento cuando se pasa una función como prop a componentes hijos.
7. useMemo
    - Memoriza valores calculados derivados de operaciones costosas.
    - Previene cálculos innecesarios en renderizados consecutivos.

III. Hooks Personalizados

Los Hooks personalizados permiten encapsular lógica común y reutilizable:

- Son funciones que usan otros hooks internamente (useState, useEffect...).
- Se nombran siempre con el prefijo use.
- Facilitan la separación de responsabilidades en componentes grandes.
- Ejemplo: useWindowSize, useFormValidation, useAuth.

Su creación responde a principios de diseño como abstracción funcional, modularidad lógica, y composición reutilizable, pilares del desarrollo web moderno.

Glosario

Término	Definición conceptual
Hook	Función que permite acceder a características de React desde componentes funcionales
Estado	Datos locales que varían en el tiempo dentro de un componente
Ciclo de vida	Fases por las que pasa un componente: montaje, actualización y desmontaje
Efecto	Operaciones secundarias que no afectan directamente el renderizado
Contexto	Mecanismo para compartir datos globales entre componentes sin prop drilling
Referencia	Acceso directo a elementos o valores sin depender del renderizado
Reductor	Función que actualiza el estado según una acción específica
Memorización	Técnica para almacenar cálculos o funciones y evitar recreaciones innecesarias

Conclusiones

Los Hooks transforman la manera de escribir y razonar sobre componentes en React, introduciendo una estructura más declarativa, modular y eficiente. Lejos de ser simplemente herramientas funcionales, representan un enfoque filosófico hacia el diseño de interfaces: claro, desacoplado, basado en la lógica y la reutilización.

Dominar sus conceptos y diferencias permite construir interfaces más limpias, escalables y fáciles de mantener, consolidando una práctica profesional en el desarrollo de aplicaciones web.