Programación IV

Profesora: Cynthia Estrada
Fecha: 21/10
Tema: *React*

Integrantes:

- Díaz Rossini Juan José
- Gallo Genaro
- Navarro Victor Leandro

---

1) ¿Qué es React?
React es una **librería de JavaScript** creada por Facebook en 2013, diseñada para construir **interfaces de usuario dinámicas y eficientes**. Su característica principal es el uso de un **DOM virtual**, lo que permite que los cambios en la interfaz se procesen rápidamente sin necesidad de actualizar todo el DOM del navegador. React no es un framework completo, sino que se centra exclusivamente en la **vista** (la capa de UI), lo que lo hace fácil de combinar con otras librerías o herramientas para manejar rutas, estados o datos.

---

2) ¿Cuál es el propósito principal de React?
El propósito central de React es **facilitar la creación de aplicaciones web interactivas y mantenibles**. Permite que la interfaz se actualice automáticamente cuando cambian los datos, evitando manipulaciones directas y repetitivas del DOM. Además, promueve la reutilización de componentes, lo que reduce la duplicación de código y mejora la consistencia visual de la aplicación. Esto es especialmente útil en proyectos grandes donde muchas partes de la UI dependen de los mismos datos o comportamientos.

---

3) ¿Qué es un componente en React?
Un componente es una **pieza independiente y reutilizable de la UI**. Representa un fragmento de la página, como un botón, un formulario, una tarjeta o incluso una sección completa. Los componentes pueden ser:

- **Funcionales:** Son funciones de JavaScript que reciben `props` y retornan JSX. Actualmente son la forma más común de escribir componentes.

- **De clase:** Clases de JavaScript que extienden de `React.Component`, usadas en versiones antiguas, que manejan estado y ciclo de vida con métodos especiales.
  Los componentes permiten dividir la aplicación en **partes pequeñas y manejables**, facilitando su mantenimiento, prueba y reutilización en diferentes contextos.

---

4) ¿Qué es JSX en React?
JSX (JavaScript XML) es una **sintaxis que mezcla JavaScript y HTML**. Permite escribir la estructura de la UI de forma declarativa, muy similar al HTML tradicional, pero con la potencia de JavaScript. Por ejemplo, podemos usar expresiones dentro de `{}` para mostrar variables, condicionales o resultados de

funciones directamente en la UI. React convierte JSX en llamadas a `React.createElement`, creando un **DOM virtual** que optimiza las actualizaciones en el navegador. Sin JSX, tendríamos que usar JavaScript puro para crear y actualizar elementos, lo que sería mucho más complejo y propenso a errores.

---

5) ¿Cómo se manejan los eventos en React?
Los eventos en React funcionan de manera similar a JavaScript, pero con diferencias importantes:

- Se usan nombres en **camelCase**, por ejemplo: `onClick`, `onChange`, `onSubmit`.
- Se asignan **funciones** como manejadores de eventos, no cadenas de texto.
- React utiliza un **sistema de eventos sintético**, que es un envoltorio que normaliza los eventos en todos los navegadores, asegurando consistencia.
  Por ejemplo, podemos manejar un click así:

```
<button onClick={() => alert("Hola!")}>Click me</button>
```

Cada vez que se dispara el evento, React asegura que la actualización del estado y la re-renderización sean eficientes.

---

6) ¿Qué es el estado en React?
El estado (`state`) es un **objeto que almacena información interna del componente**, que puede cambiar a lo largo del tiempo y afectar cómo se renderiza la UI. Por ejemplo, un contador, un formulario que guarda datos o la visibilidad de un modal. A diferencia de las `props`, que son **inmutables y vienen de fuera**, el estado es **propio del componente** y permite que la UI reaccione a cambios internos de manera automática.

---

7) ¿Cómo se actualiza el estado en React?
El estado **nunca se debe modificar directamente**, porque React necesita detectar los cambios para re-renderizar el componente. Para actualizarlo se usan:

- En **componentes de clase:** `this.setState({ clave: nuevoValor })`.
- En **componentes funcionales:** el hook `useState` devuelve una función para actualizar el estado:

```
const [contador, setContador] = useState(0);
setContador(contador + 1);
```

Cada vez que se llama a la función de actualización, React vuelve a renderizar solo las partes necesarias de la UI.

---

8) ¿Qué es un hook en React?
Un hook es una **función especial que permite usar características de React como estado, efectos y contexto** dentro de componentes funcionales. Antes de los hooks, solo los componentes de clase podían tener estado y ciclo de vida. Los hooks más comunes son:

- `useState`: manejar estado local.
- `useEffect`: realizar efectos secundarios como peticiones HTTP o suscripciones.
- `useContext`: consumir valores de un contexto global.
- `useMemo` y `useCallback`: optimización de rendimiento.
  Los hooks permiten escribir componentes **más simples, legibles y reutilizables** sin necesidad de clases.

---

9) ¿Qué es el contexto en React?
El contexto es una forma de **compartir información global** entre componentes sin tener que pasar `props` manualmente a cada nivel del árbol. Se usa para temas, configuraciones, datos de usuario, idiomas, etc.
Se crea con `React.createContext()`, se provee con `<Context.Provider value={...}>` y se consume con `useContext(Context)`. Esto simplifica mucho la gestión de datos que deben estar disponibles en toda la aplicación y evita el **"prop drilling"** (pasar props innecesarios por varios niveles).

---

10) ¿Cómo se optimiza el rendimiento en React?
Algunas estrategias para mejorar el rendimiento son:

- **Memoización de componentes:** con `React.memo` para evitar renders innecesarios.
- **Memoización de funciones y valores:** usando `useCallback` y `useMemo` para que no se recalculen en cada render.
- **Key únicas en listas:** para que React identifique correctamente elementos al renderizar listas y evite re-renderizaciones completas.
- **División de componentes grandes:** en componentes más pequeños y específicos, mejorando la eficiencia.
- **Evitar lógica pesada en render:** mover cálculos complejos a funciones externas o hooks.
- **Lazy loading y code splitting:** cargar solo lo necesario, reduciendo el tamaño inicial de la app.