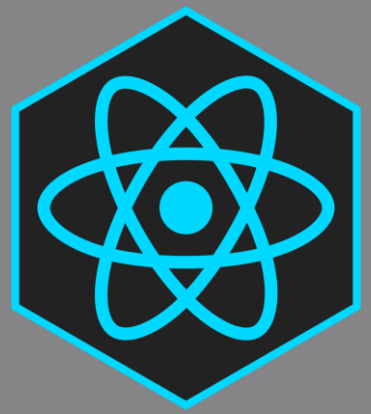


Parte 2: Operaciones básicas con React: Virtual DOM, Propiedades y hooks

UD4: Fundamentos de React

Después de este documento...

- Seguirás conociendo conceptos fundamentales de React como el **Virtual DOM** y el renderizado de componentes
- Sabrás cómo pasar **propiedades** a los componentes de React, lo que permitirá que sean "personalizables"
- Veremos cómo usar **eventos**
- Sabrás lo que es un **Hook** y habrás usado el fundamental: **useState**

DOM "real"

- DOM es la representación en forma de árbol de la interfaz gráfica de nuestra aplicación
  - Si cambia el estado, cambia esta interfaz para incluir las modificaciones
  - Como es un árbol, todos los hijos del nodo cambiado tienen que ser pintados (hayan cambiado o no)
  - Actualizar el DOM es un tarea costosa en cuanto a rendimiento se refiere
    - Más cambios, más lenta la web

Virtual DOM

- Representación en **memoria** del DOM real que actúa de intermediario entre el estado de la aplicación y el DOM
- En React, cada pieza de la UI es un componente y cada componente posee un **estado** interno
  - Este estado es observado por React para detectar cambios
    - Si suceden se actualiza el Virtual DOM y sigue el mismo proceso para trasladar los cambios a la interfaz presentada en el navegador

Renderizado

```
function SaludarPersona(){
const usuario = {id: 1, nombre:
"María", apellido: "Rodríguez"}
return
<h1>{JSON.stringify(usuario)}</h1>
}
```

```
return (
    <div key={usuario.id}>
      <h1>{usuario.nombre}</h1>
      <h3>{usuario.apellido}</h3>
    </div>
);
```

La propiedad key la usa React para su funcionamiento interno (actualizaciones)

```
ReactDOM.render(<SaludarPersona />,
document.getElementById("root"));
```

En una línea

- Muestra:

{"nombre":"María","apellido":"Rodríguez"}

- Puedo cambiar el return para especificar los campos que quiero mostrar

María

Rodríguez

1. El render de ReactDOM llama al componente <SaludarPersona />
2. El componente devuelve el código HTML (sustituyendo las expresiones por su valor)
3. ReactDOM actualiza el DOM

EJERCICIO PROPUESTO I: Listar usuarios

- Crea un componente que procese un array de usuarios y cree una lista html
  - Para ello puedes usar un método como **map**

```
const usuarios = [
  { nombre: "Manolito", id: 1 },
  { nombre: "Antonia", id: 2 },
  { nombre: "Leopoldo", id: 3 }
];
```

Nombres de usuario

- Manolito
- Antonia
- Leopoldo

Propiedades

- Se reciben como **parámetro** por valor en la función del componente
  - En el JSX son atributos
  - En el componente son propiedades de un objeto literal que puede ser desestructurado pero NO modificado
- Estos códigos hacen lo mismo:
  - Arriba es un objeto y abajo es un string (valor de la propiedad)

```
root.render(
  <div>
    <Mostrar title="Soy Manolito" />
    <Mostrar title="Soy Leopoldo" />
  </div>
);
```

Soy Manolito

Soy Leopoldo

```
export function Mostrar(props){
    return <h1>{props.title}</h1>
}
```

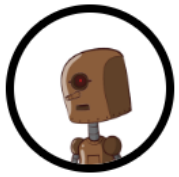
```
export function Mostrar({title}){
    return <h1>{title}</h1>
}
```

```
const Mostrar = ({title})=>
  <h1>{title}</h1>
```

Con función flecha

EJERCICIO PROPUESTO II: User Card

- Crea un componente llamado **UserCard** que reciba los siguientes parámetros:
  - **name**: cadena
  - **amount**: entero
    - Si no se pasa un valor, por defecto será 0
  - **worker**: booleano
  - **points**: array de reales
  - **address**: objeto formado por dos campos de tipo cadena - address y city
  - **avatar**: url a una imagen alojada en un servidor
    - Puedes usar robohash para generarla
  - **greet**: función que saluda
    - Observa que hasta se pueden pasar funciones
- Muestra varios componentes **UserCard** con datos de ejemplo



Antonia Jiménez

Trabaja por 1000€

Ciudad:NY
Calle:123 Main Street

```
<div>
   <div>
      <img />
   </div>

   <h5></h5>

   <p></p>

   <ul>
      <li></li>
      <li></li>
   </ul>
</div>
```

Eventos en React

- Son eventos sintéticos
  - Funcionan de forma similar a los que conocemos pero no son eventos reales del DOM, son una capa de React
- React usa mucho las definiciones en línea:
  - Nombre del evento con "on" delante, sintaxis **camelCase**:

```
<input id="texto" onChange={(e) => {
      console.log('Valor en el input: '+e.target.value)
  }}/>
```

- También se puede sacar la función manejadora:
  - Código más limpio

```
const manejaChange = (e) => {
  console.log('Valor en el input: '+e.target.value)
}
root.render(<> <input id="texto" onChange={manejaChange}/> </>);
```

Tipos de eventos

- Como recordatorio, los más usados son:
  - Escuchadores de eventos de ratón:
    - **onClick**: botón izquierdo del ratón
    - **onMouseOver**: pasar el ratón sobre un elemento
    - **onMouseOut**: sacar el ratón del elemento
  - Escuchadores de eventos de teclado:
    - **onKeyPress**: pulsar una tecla
  - Escuchadores de eventos sobre elementos:
    - **onFocus**: poner el foco en un elemento (ej. <input>)
    - **onBlur**: quitar el foco de un elemento
    - **onChange**: al cambiar el contenido de un <input>,<select>…

EJERCICIO PROPUESTO III: Cheese Hater (1/2)

- Crea un componente **CheeseHater** que conste de un textarea de forma que al cambiar de valor comprueba si se ha escrito la palabra “queso” en cuyo caso muestra un alert diciendo 'ODIO EL QUESO’

```
const manejadora = (evento)=>{
  if (evento.target.value.toLowerCase().includes("queso") )
    alert ("Odio el queso");
    }

const CheeseHater = ()=><textarea onChange={manejadora}></textarea>
```

Estado de los componentes

- Hay componentes **sin estado** que no guardan ninguna información (los realizados hasta ahora)
- Los componentes pueden tener un **estado**, que se utiliza para almacenar información sobre el componente que puede cambiar con el tiempo
  - Similar a las **props,** pero **privado**, y controlado por el componente
  - El componente **reacciona al cambio de estado**, actualizando la vista cuando sea necesario

Hooks

- Los **hooks** fueron introducidos en React en la versión 16.8 (2019)
- Son funciones que nos permiten conectarnos al estado de React
  - Los más usados son:
    - **useState**
    - **useEffect**
    - useContext
  - Se definen dentro del componente, antes del return

useState

- **UseState** es un Hook que permite definir el estado y modificarlo
  - Para usarlo hay que importarlo

```
import { useState } from "react";
```

  - Permite definir su **valor** inicial
  - Se le puede pasar la función **set** encargada de modificar ese dato

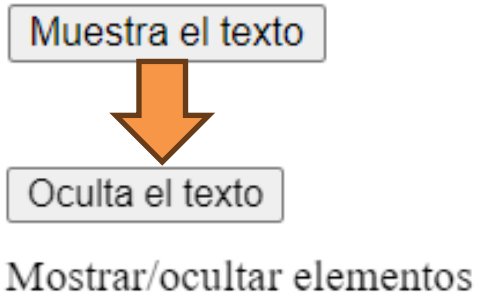
```
const [nombre, setNombre] = useState ("Hermione");
```

  - Se accede como a una variable "normal"

```
<p>{nombre}</p>
```

Ejemplo: Mostrar/ocultar texto

- Div que se muestra/oculta según un estado "show" que va alternando en cada click de botón



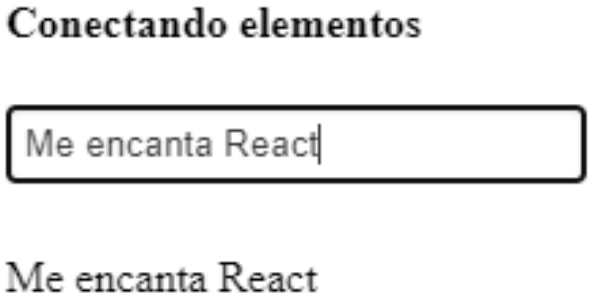

```
function MostrarOcultar() {
  const [show, setShow] = useState(true);

  return (
    <>
      <button onClick={() => setShow(!show)}>
        {show ? "Oculta el texto" : "Muestra el texto"}
      </button>
      {show?<div>Mostrar/ocultar elementos</div>:null}
    </>
  );
}
```

EJERCICIO PROPUESTO III: Cheese Hater (2/2)

- Modificar **CheeseHater** para que además del alert, cambie el color de fondo a rojo
  - Estilos condicionales
  - Necesidad de variable booleana para tomar la decisión

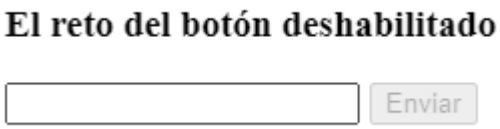
EJERCICIO PROPUESTO IV: Vinculando elementos

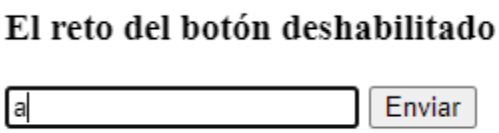
- Crea un componente con un campo de texto de forma que todo lo que se escriba en el mismo se traslade como un eco a un párrafo situado a continuación



EJERCICIO PROPUESTO V: El botón que se deshabilita

- Crea un componente con un campo de texto y un botón de forma que el botón esté habilitado/deshabilitado en función de si hay algo o no en el campo de texto





EJERCICIO PROPUESTO VI: Contador

- Implementar un contador que se incremente o decremente en función de lo que el usuario decida pulsando un botón
  - El color del valor es diferente según sea negativo, positivo o 0
- Pasos:
  1. Declarar un estado para el valor del contador
  2. Incrementar o decrementar el valor a través de setState().
  3. Hacer que los botones HTML escuchen al evento onClick y generen código JSX