



**¡Les damos la
bienvenida!**

¿Comenzamos?

Esta clase va a ser

- grabada



Encuesta After Class

Por encuestas de Zoom

¡Coordinamos nuestro After Class!

Clase 12. REACT JS

Renderizado Condicional

Objetivos de la clase

- **Profundizar** sobre renderizado condicional y sus implicancias.
- **Diagnosticar** y solucionar problemas de *rendering*.

MAPA DE CONCEPTOS



Temario

11

Context

- ✓ Contexto
- ✓ Contexto dinámico
- ✓ Nodo proveedor
- ✓ Custom Provider

12

Técnicas de Rendering

- ✓ Recapitulando: principios básicos react
- ✓ Rendering condicional
- ✓ Render optimization

13

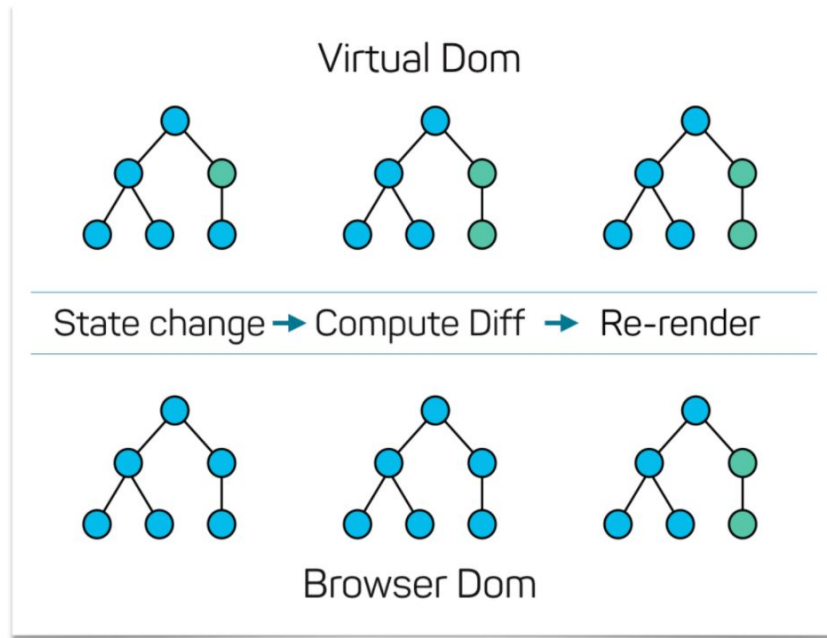
Firebase I

- ✓ Firebase
- ✓ ¿Por dónde empiezo?
- ✓ Firestore
- ✓ Configurando nuestra app

Recapitulando: Principios básicos React

Como vimos en las primeras clases, React trabaja con un **flujo de reconciliación** y entiende bastante acerca de dónde y cómo ocurren los cambios en nuestra app.

A su vez mantiene una versión en virtual de la misma y recordamos eso como el virtual DOM

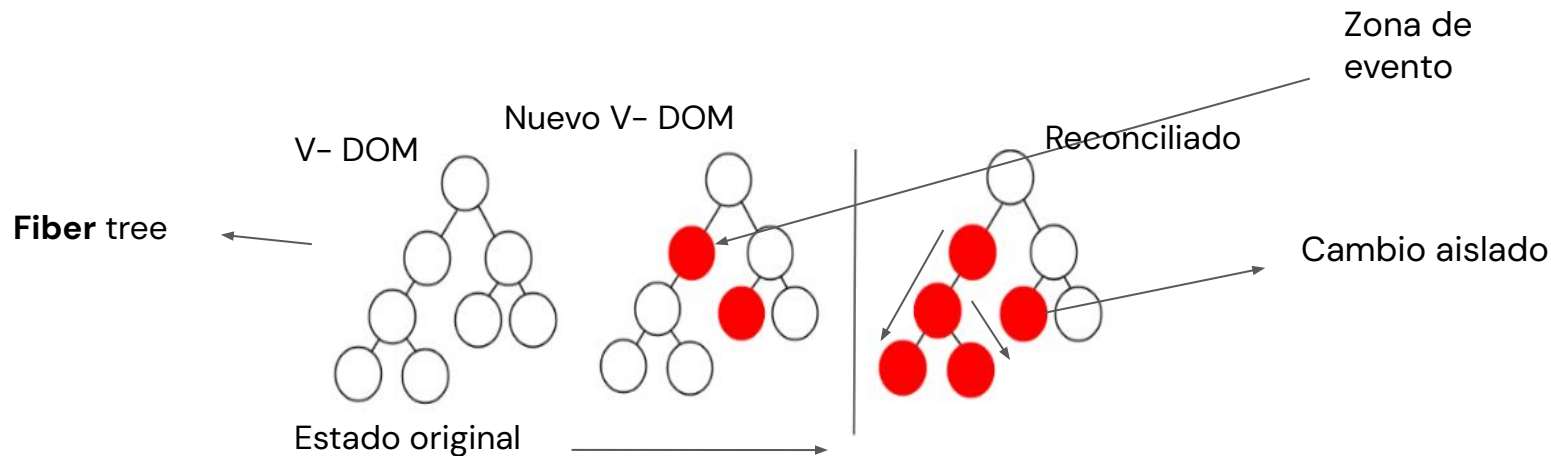


Si bien como vimos anteriormente, React nos ayuda a escuchar eventos, nos sigue dejando a nuestro cargo la sincronización con nuestro estado.

```
const ControlledInput = () => {  
  const [input, setInput] = useState('');  
  
  return <input type="text"  
    value={value}  
    onChange={(evt) => setInput(evt.target.value)} />;  
}
```

El cada nuevo render causado por `setInput` reescribirá un `input` con el nuevo **value**

Recordemos la reconciliación



Rendering Condicional

Rendering Condicional

En ciclos de render puedo decidir que quiero hacer rendering de sólo algunos nodos de un árbol, o de otro:

```
function LoadingComponent() {  
  const [loading, setLoading] = useState(true);  
  
  useEffect(() => {  
    setTimeout(() => {  
      setLoading(false);  
    }, 10000);  
  }, []);  
  
  return <>  
    { loading ? <h2>Loading</h2> : <h3>Loaded!</h3> }  
  </>  
}  
  
export default function App() {  
  return <LoadingComponent />  
}
```

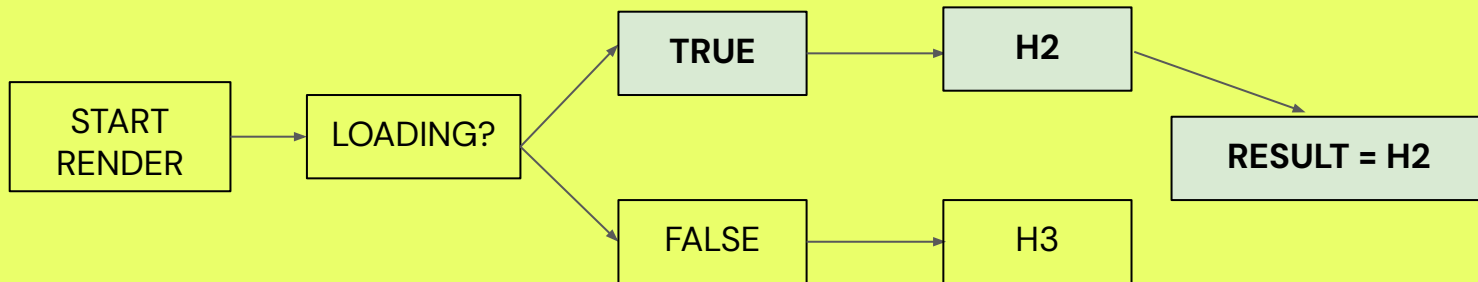
Loading

Console

React maneja el asunto con comodidad

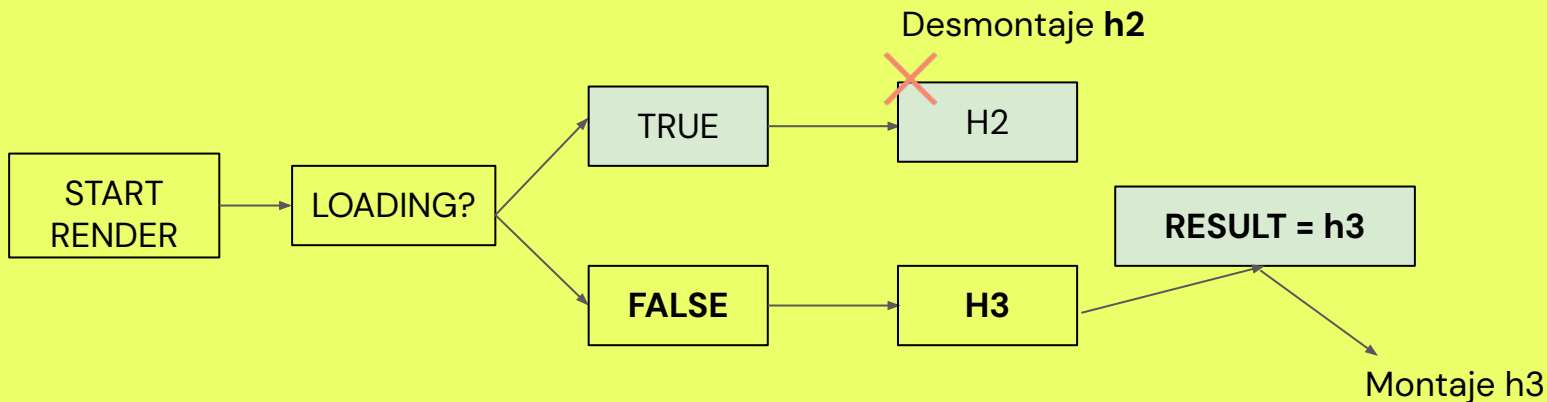
Rendering Condicional

Si pensamos en React- way obtendremos este diagrama:



Rendering Condicional

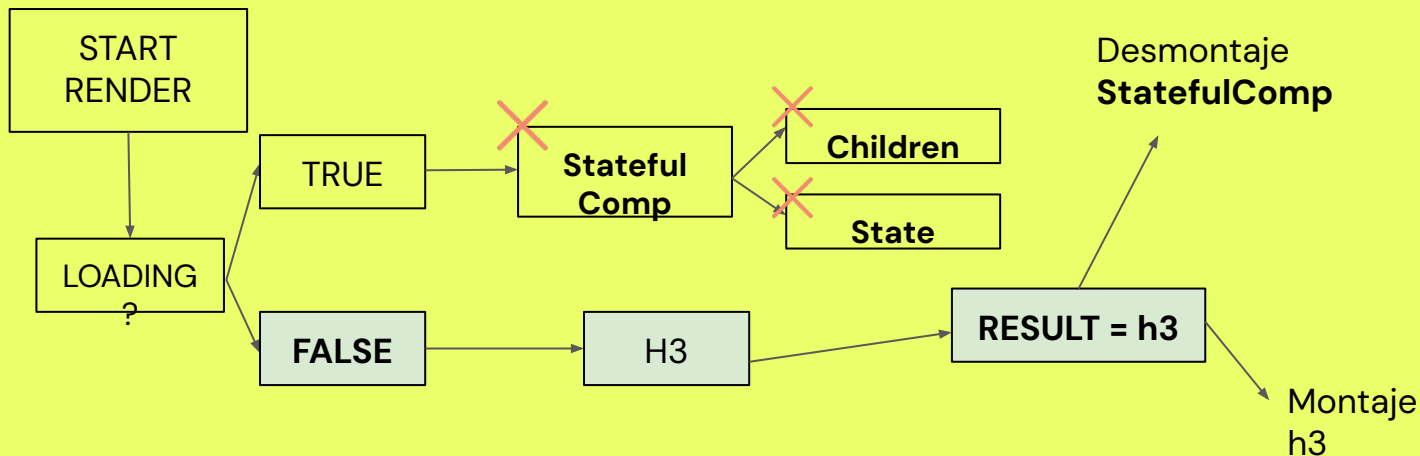
El segundo render se vería así:



El caso es simple porque un h2 o h3 no son componentes complejos que tengan una sub- jerarquía de nodos dentro.

Rendering Condicional

Si en cambio...



Si por algún motivo volviese a hacer trigger de loading, h3 sufriría el mismo destino de ser desmontado, sin importar cuantos children tuviera.

Rendering condicional

- ✓ Sirve para aparecer y desaparecer nodos del render.
- ✓ Estos eventos provocan dismounting y todos los efectos que ellos conlleva.
- ✓ Se llamará efecto de desmontaje y podremos detectarlo.
- ✓ Podemos usar los cleanup effects para detectar algún dismounting si no sabemos con certeza si ocurre.
- ✓ A veces se producen sin intención y **causan bugs** o pérdida no intencionada del estado, dando inestabilidad.



Ejemplo en vivo

Vamos al código

Técnicas de Rendering Condicional

Rendering Condicional #1

IF c/ return temprano

React renderiza el resultado del return de nuestra función y cada return se transforma en nuestro nuevo árbol de partida para próximos.

```
function TextComponent({ condition }) {  
  
  if(condition === true) {  
    return <h2>Condition is true</h2>  
  }  
  
  return <h2>Condition is false</h2>;  
}
```

Versión JSX

vs

```
function TextComponent(_ref) {  
  var condition = _ref.condition;  
  
  if (condition === true) {  
    return React.createElement("h2", null, "Condition is true");  
  }  
  
  return React.createElement("h2", null, "Condition is false");  
}
```

¡Notar que se crean 2 **createElement** distintos!

Vanilla JS (Después de ser transpilado)

Rendering Condicional #2

Inline con fragment

Mantenemos el mismo nodo como padre y modificamos sus children, que en este caso son los textos, lo cual optimiza ya que no hay dismounts:

```
function TextComponent({ condition }) {  
  return (<>  
    {condition && <h2>Condition is true</h2>}  
    {!condition && <h2>Condition is false</h2>}  
  </>);  
}
```

Versión JSX

vs

```
function TextComponent(_ref) {  
  var condition = _ref.condition;  
  return React.createElement(React.Fragment, null, condition && React.createElement("h2",  
null, "Condition is true"), !condition && React.createElement("h2", null, "Condition is  
false"));  
}
```

Vanilla JS

¡Notar que se crean 2 **createElement** distintos!

Rendering Condicional #3

Inline ternary

Mantenemos el mismo nodo como padre y modificamos sus children, que en este caso son los textos, lo cual optimiza ya que no hay dismounts:

```
function TextComponent({ condition }) {  
  return <h2>{condition === true ? 'Condition is true' :  
    'Condition is false'}</h2>;  
}
```

Versión JSX

vs

```
function TextComponent(_ref) {  
  var condition = _ref.condition;  
  return React.createElement("h2", null, condition === true ? 'Condition is true' :  
    'Condition is false');  
}
```

Vanilla JS

¡Notar que se crea un único createElement!

Rending Condicional-Bonus!

```
const InputCount = ({onConfirm, maxQuantity}) => {  
  };  
const ButtonCount = ({onConfirm, maxQuantity}) => {  
  };  
  
export default function ItemDetail({ item, inputType = 'input' }) {  
  const Count = inputType === 'button' ?  
    | ButtonCount : InputCount;  
  const itemMax = item.max;  
  const min = item.min;  
  
  function addToCart(quantity) {  
    if(item.inStock) {  
      console.log(`Agregar al cart el item: ${item.id}  
        | con cantidad: ${quantity}`);  
    }  
  }  
  
  return (  
    <div>  
      <label>Item description </label>  
      <Count onConfirm={addToCart} maxQuantity={item.max}></Count>  
    </div>  
  );  
}
```

Retomando el ejemplo de la clase 9

¿Preguntas?



Crea un Loader Component

Crea en [stackblitz](https://stackblitz.com) un componente “Loader” dentro de la app, que tenga una prop “loading” (boolean:true|false).

Duración: **15 minutos**



ACTIVIDAD EN CLASE

Crea un Loader Component

Descripción de la actividad.

Crea en [stackblitz](https://stackblitz.com) un componente "Loader" dentro de la app que tenga una prop "loading" (boolean:true|false). Si loading es true, el componente debe mostrar "Loading..." y si es false, nada.

Extra: si quieres, puedes integrar algún spinner de la librería de UI que estés usando.



Break

¡10 minutos y volvemos!

Otras técnicas para control condicional

Conditional props: styling

El conditional rendering aplica no sólo para los nodos, sino también para sus propiedades.

```
function TextComponent({ condition }) {  
  return (<  
    <h2 style={{ color: !condition ? 'red' : 'green' }}>Loading...</h2>  
    </>);  
}  
  
export default function App() {  
  return <TextComponent condition={true} />  
}
```

Loading...

¡Notar que se crea un único **createElement**!

Conditional Attributes: classes

Modificar clases en base a condiciones:


```
function TextComponent({ condition }) {  
  return (<  
    <h2 className={condition === true ? 'greenClass' : 'redClass'}  
    >Loading...</h2>  
    </>);  
}
```

```
function TextComponent(_ref) {  
  var condition = _ref.condition;  
  return  
    React.createElement(React.Fragment, null, React.createElement("h2", {  
      className: condition ? 'greenClass' : 'redClass'  
    }, "Loading..."));  
}
```

Como vemos, ¡las clases se concatenan!

Conditional Attributes: multi- class

```
function TextComponent({ condition, other }) {  
  return (<  
    <h2 className={`${condition === true ? 'redClass' : 'greenClass'} ${other || ''}`}>Loading...</h2>  
    </>);  
}  
  
export default function App() {  
  return <TextComponent condition={true} other="newClass" />  
}
```




▼<div id="root">
 <h2 class="redClass newClass">Loading...</h2> ==
</div>

Conditional Attributes: multi- class anti-pattern

En este caso, no es conveniente aplicar **condition &&**.

```
function TextComponent({ condition, other }) {  
  return (<  
    <h2 className={`${condition === true ? 'redClass' : 'greenClass'} ${other && 'otherClass'}`}  
    >Loading...</h2>  
  </>);  
}  
  
export default function App() {  
  return <TextComponent condition={true} />  
}
```



```
<body>  
  <div id="root">  
    <h2 class="redClass undefined">Loading...</h2> == $0  
  </div>  
</body>
```


Conditional props/props dinámicas

Podemos hacer spreading de propiedades de manera condicional:

```
function TextComponent({ condition, other }) {  
  const config = condition ? {  
    className: `redClass ${other || ''}`,  
    title: 'Title when condition is true'  
  } : {};  
  
  return (<  
    <h2 {...config}>Loading...</h2>  
  </>);  
}  
  
export default function App() {  
  return <>  
    <TextComponent condition={true} />  
  </>  
}
```

```
<body>  
  <div id="root">  
    <h2 class="redClass " title="Title when condition is true">  
      Loading...</h2> == $0  
    </div>  
  </body>
```

Loading...

Console

Preview (local)

Console was cleared



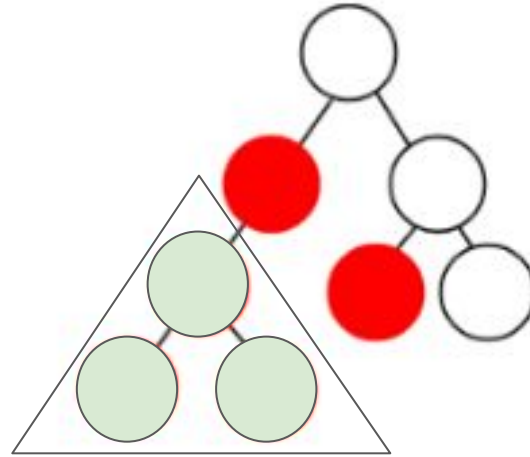
Ejemplo en vivo

Vamos al código

Render Optimization

Esto quiere decir que **podemos salvarle** a React ese trabajo, si:

- ✓ El componente es **puro**.
- ✓ Tenemos la certeza de que **las mismas props producen siempre el mismo render**.
- ✓ Sabemos que es muy caro de realizar, una lista larga, compleja, etcétera.



Memoizing

Memo (Ization)

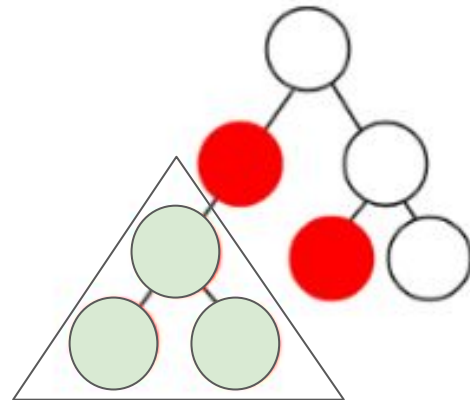
Para lograrlo, solo necesito envolver mi componente en un memo:

```
function Comp() { return <> }
```

```
React.memo(Comp, fn*)
```

Donde `fn` es la función comparadora

```
*fn = (prevProps, nextProps) => true|false  
true => usar memo! / false => re-render
```



Memoizing

React Memo: Configuración default

Por default se usa comparación superficial (shallow):

```
const ListItem = React.memo(({ item }) => {  
  |   return <li>{item.id}</li>  
  | })
```

Previo: { id: 1, name: 'item 1 name' }

Nuevo: { id: 1, name: 'item 1 name' }

Solo se invocará el render al montar

React Memo: validación manual

Puedo decirle a React que sólo haga re- rendering cuando una propiedad específica cambie:

```
const ListItem = React.memo(({ item }) => {  
  console.log('Rendering item');  
  return <li>{item.id}</li>  
}, (oldProps, newProps) => oldProps.item.modifyDate === newProps.item.modifyDate)
```

En este caso, solo cambiará cuando **modifyDate** sea distinta

React Memo

- ✓ Sirve para ahorrar renders costosos de los cuales podamos prever el resultado en base al análisis de sus props.
- ✓ No es necesario **ni recomendado** usarlo en todos lados o en componentes simples.
- ✓ Útil en listas largas y determinadas que se tienen un re-rendering frecuente pero que no modifica sus props.
- ✓ Si te interesa expandir, ¡también hay un hook para [memoizar](#) otros cálculos aparte de renders!



#Codelalert

Ingresa al manual de prácticas y realiza la sexta actividad “Cart View”. Ten en cuenta que el desarrollo de la misma será importante para la resolución del Proyecto Final.



Cart View

Descripción de la actividad.

- ✓ Expande tu componente Cart.js con el desglose de la compra y actualiza tu CartWidget.js para hacerlo reactivo al contexto

Recomendaciones

- ✓ Cart.js
- ✓ Debe mostrar el desglose de tu carrito y el precio total.
- ✓ Debe estar agregada la ruta 'cart' al BrowserRouter.
- ✓ Debe mostrar todos los ítems agregados agrupados.
- ✓ Por cada tipo de ítem, incluye un control para eliminar ítems.
- ✓ De no haber ítems muestra un mensaje, de manera condicional, diciendo que no hay ítems y un react-router Link o un botón para que pueda volver al Landing (ItemDetailContainer.js) para buscar y comprar algo.



Cart View

Recomendaciones

- ✓ CartWidget.js.
- ✓ Ahora debe consumir el CartContext y mostrar en tiempo real (aparte del ícono) qué cantidad de ítems están agregados (2 camisas y 1 gorro equivaldrían a 3 items).
- ✓ El cart widget no se debe mostrar más si no hay items en el carrito, aplicando la técnica que elijas (dismount, style, etc).
- ✓ Cuando el estado interno de ItemDetail tenga la cantidad de ítems solicitados mostrar en su lugar un botón que diga "Terminar mi compra"

¿Preguntas?

Resumen de la clase hoy

- ✓ Rendering condicional.
- ✓ Memoization.
- ✓ Estrategias de optimización.

Opina y valora
esta clase

Muchas gracias.