

Dinamizar reacciones para las acciones

"When making experiences, attention to detail matters."

Richard Branson – Founder of Virgin Group.

Estados

Las aplicaciones web dinámicas, (en algunos casos llamadas [Single Page Application, o SPA](#)) tienen la habilidad de cambiar partes de su interfaz gráfica durante su uso, como una reacción a una acción propiciada por una persona.

Por ejemplo, la aplicación que hemos venido construyendo en las meetings anteriores, muestra una lista de álbumes musicales. Cada álbum es un componente funcional sin estado, pero ¿qué pasaría si decidiéramos ofrecer la opción de **amar** a un álbum, a través del click de un icono de corazón?



Estos comportamientos dinámicos, que modifican la interfaz gráfica de un componente, se logran a través del [manejo de estados](#), el cual es el tema que tocaremos en esta toolbox, y en la próxima meeting.

En desarrollo web, y a nivel conceptual, **el estado es un valor propio de un componente en un momento determinado.**

Pero, antes de indagar técnicamente en el concepto de estados en una aplicación web, especialmente con React, vamos a asimilar este concepto de forma metafórica utilizando aspectos de nuestra vida real. Como te hemos dicho antes, los componentes de React y las personas, tienen mucho en común.



El componente <Persona />

En estos momentos estás en reposo, quizás en una silla, leyendo esta toolbox. ¿Cómo describirías tu estado actual? ¿Qué características podrías decir que definen tu estado? Haz una pausa y construye una lista de variables que contengan todas las características que consideres que forman parte de tu estado actual, y asígnales un valor a cada una, por ejemplo `let durmiendo = false;`. ¡Esperemos no estés durmiendo!

En una persona, el **estado** puede tener distintas formas, valores y significados. Sentirse bien, con ánimo o con hambre, son estados emocionales, mientras que estar caminando, saltando o subiendo una escalera, son estados físicos. ¿Puedes pensar en otras categorías de estados? Te dejamos una lista de ejemplos de estados para un hipotético componente `<Persona />`.

...

// algunos ejemplos de estados para el componente <Persona />

```
let durmiendo = false;
let comiendo = true;
let enMovimiento = false;
let feliz = true;
let triste = false;
let ansioso = true;
let edad = 35;
let ubicacion = "mi casa";
let nivelConcentracion = 400;
let estadoCivil = "soltero";
let amigos = ["Phoebe", "Joey", "Chandler", "Monica", "Rachel", "Ross"];
...
```

Como te comentamos anteriormente, **el estado es un valor propio de un componente en un momento determinado**. Los ejemplos que te mostramos en el gist, son valores de variables que pueden pertenecer a una persona y que **pueden cambiar en el tiempo**, por un evento determinado. Por ejemplo, podemos cambiar el valor de la variable 'ubicación' si nos movemos de un lugar a



otro, o podemos aumentar nuestro 'nivelConcentracion' si apagamos las notificaciones del celu.

Estados y props

Es importante entender la diferencia entre lo qué es y no es un estado. Por ejemplo, el color de ojos, nuestros nombres, el número de identificación gubernamental, no son estados, sino propiedades. ¿Recuerdas las **props** de los componentes de React? Mismo concepto.



A diferencia del **estado**, las propiedades, o **props**, son características de una entidad que son especificadas por otra entidad. Por ejemplo, tu número de identificación te fue asignado cuando te registraste en tu ciudad, y tu color de ojos fue creado y asignado debido a características genéticas de tus progenitores. Si tu fueras un componente de React, podrías tener esta implementación.

...

```
<Persona colorOjos="marron" id={123456789} nombre="Jesi" />
```

...

El estado es una variable interna que existe en el componente. Sin embargo, el estado puede inicializarse a través de una propiedad. Por ejemplo, te habíamos mostrado en el gist que tu ubicación actual era parte de tu estado, pero ese valor de 'mi casa' pudo haber sido asignado como una propiedad inicial, y luego cada



vez que te muevas a otros lugares, el valor de la variable 'ubicación' irá cambiando respectivamente.

Los estados pueden cambiar a lo largo del tiempo, pero es la entidad que lo posee la responsable de modificarlo, mientras que una prop tiende a ser algo que se mantiene, y para ser actualizada, vamos a requerir de valores u operaciones externas al componente.

Por ejemplo, si pudieras cambiar el color de tus ojos con unos lentes de contacto, esto implicaría depender de procesos externos a ti, es decir, que los lentes estén a la venta, que tengan un precio accesible, etc. Pero, si estuvieras corriendo, en un estado activo, y te quieres detener, será producto de una decisión propia, a menos que te choques con una pared.



La diferencia entre estados y props es una conversación en escala de grises, no está escrito en piedra lo que es una prop o un estado, pero, en el frontend, poco a poco vamos a empezar a reconocer patrones que se irán repitiendo lo suficiente como para poder decidir lo que es un estado o una prop. Esto lo vamos a ver a estudiar desde hoy, hasta que finalices la cursada.

Ok, buenísimo, pero... ¿y React?

Que bueno que te estés preguntando eso, porque justo vamos a llevar nuestra metáfora de las personas, sus estados y propiedades a React-landia. El estado, además de ser un concepto teórico en el desarrollo web, tiene su aplicación práctica en React, de hecho, el estado es el concepto transversal a todos los tópicos de esta librería.

Una aplicación web es una gran [máquina de estados](#) cuyos valores van cambiando a medida que recibe impulsos de personas usuarias, enmarcados en la activación de eventos. A continuación, te presentamos la aplicación más



sencilla del mundo: un checkbox que le permite a las personas marcarlo o desmarcarlo. Este componente '`<Checkbox />`' lo único que hace es renderizar un input de tipo 'checkbox'.



Este componente tiene solo dos estados: chequeado y no chequeado, pero si empezamos a pensar como developers, podemos definir una variable que indique **el nombre del estado** que se desea modificar, junto con **un valor**. En este caso, como el estado solo tiene dos opciones (es decir, es binario) podemos declarar una variable 'chequeado' cuyo valor sea 'true' o 'false' dependiendo del caso. Recuerden, **el estado es el valor de una variable de un componente en un momento específico**.

Para el caso particular de un checkbox, HTML nos da controles gratuitos para poder alterar el valor de su estado a través de un click, pero, en el contexto de una aplicación con React, nosotros/as vamos a querer controlar por completo el comportamiento de este componente. Presta atención al siguiente video para ver los problemas con los que nos podemos encontrar si decidimos implementar un checkbox en HTML con React.

[Estados del checkbox sin React](#)

Manejando estados con React

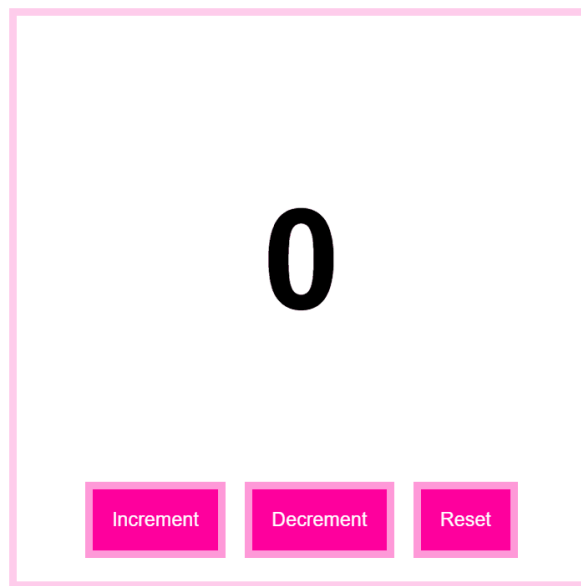
Como te mencionamos anteriormente, el concepto de estados es transversal a todos los conceptos que aprenderás de React. Un componente puede tener uno o varios estados, cuyos valores serán cualquiera de los permitidos por Javascript.

- strings
- números
- booleanos
- arrays
- objetos
- funciones



- null
- undefined

Una aplicación web dinámica podrá tener distintos tipos de estados representados por distintos tipos de variables, y dependiendo del caso, algunos componentes deberán saber el estado de otros componentes para tomar algún tipo de decisión. Como developers, tendremos la potestad de decidir qué tipo de valor va a corresponder con un tipo de estados. Para un checkbox, podemos usar un booleano, pero, ¿qué tipo de variable usarías para manejar el estado en una aplicación en la que se desee incrementar, decrementar o resetear un valor?



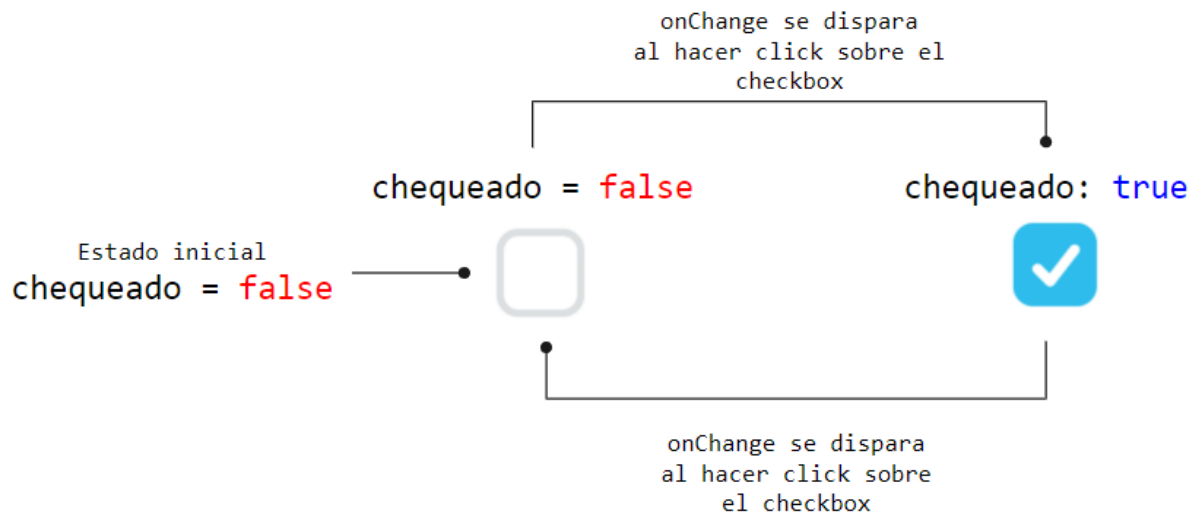
Algo más que deberías saber de los estados es que existen dos escalas: locales y globales. El estado local son los valores que son importantes para un solo componente, y a su vez son administrados por el mismo. Por ejemplo, el componente `<Checkbox />` que te mostramos antes se encargará de administrar su propio estado. El estado global son aquellos valores que son importantes para varios componentes al mismo tiempo. Por ahora, nos vamos a enfocar en los [estados locales](#).

El manejo de estados consiste en programar los mecanismos para actualizar los valores en uno, o varios componentes, así como setear su valor inicial. Por ejemplo, para manejar el estado del checkbox deberemos:

- Setear el valor inicial entre 'true' o 'false'
- Cambiar el valor actual del checkbox al dispararse el evento 'onChange'



El siguiente diagrama de estados nos indica el ciclo que sigue este componente, según los eventos programados:



Cuando la aplicación se carga por primera vez, se ha decidido que el estado inicial sea 'false' para el checkbox, y se espera que cuando el evento 'onChange' se dispare, ese estado cambie a 'true'. La situación inversa deberá ocurrir cuando el evento 'onChange' se dispare de nuevo.

React hooks

Existen dos formas de manejar estados dentro de componentes de React: [con clases de Javascript](#) y con una funcionalidad de React llamada "hooks". Este nombre (hooks) se da porque son funcionalidades que se "enganchan" a los componentes, pero honestamente, es más un concepto de marketing. El hook en sí es una función de Javascript que ofrece ayuda para solventar un problema en específico. **En este programa vamos a hacer énfasis en los hooks por su gran facilidad de uso y su aceptación dentro de la comunidad frontend de React.**

En React, existen varios tipos de hooks que solventan distintos tipos de problemas, y hoy te hablaremos del primero: 'useState()'. Este hook te permite manejar estados dentro de componentes funcionales.

El hook es una función (como las de Javascript) que provee la librería React y para poder usarla, deberás importarla en el archivo donde se encuentre el, o los componentes que tendrán estados. Esta función nos permite **definir el valor**



inicial de un estado, y a su vez **retorna el valor del estado actual así como el método de su actualización**.

El valor inicial de un estado lo definimos como el argumento de la función, mientras que el valor actual y el método de actualización son retornados por la función en un array que siempre tendrá dos posiciones. En la posición [0] del array está el valor del estado actual, y en la [1] la función de actualización.

(Valor del estado actual) (Función para actualizar el estado actual)
Index 0 Index 1

```
let infoDelEstado = useState(estadoInicial);
```

¿Qué te parece si ponemos estos conceptos en práctica?

[Manejando estados con useState hook](#)

Nota: encontrarás el sandbox en la descripción del video.

Estados y re-renders

Al principio de la toolbox te comentamos que las aplicaciones web dinámicas tenían la habilidad de poder cambiar partes de su interfaz gráfica en base a una interacción. Te mostramos el ejemplo de la acción de “amar un álbum” presionando sobre el icono del corazón.





Si el componente tiene el corazón vacío, podemos decir que tiene un estado 'liked' seteado en 'false', y en el caso contrario, es decir, con el corazón en rojo, el estado 'liked' estaría en 'true'. Este cambio de interfaz se da cuando se hace click sobre el área del corazón, aplicando una lógica similar al del checkbox, pero con dos svg's. ¿Cómo maneja React este cambio gráfico? A través del Virtual DOM.

Cada vez que se actualiza el estado en un componente, React actualiza la UI. Un cambio de estado en un componente representa el cambio de una variable, de un valor a otro. Cuando eso ocurre, React decide qué partes de la interfaz gráfica actualizar comparando el Virtual DOM con el DOM actual, de esta forma inserta el nuevo HTML que se necesita mostrar, mostrando la nueva interfaz gráfica.

Esto hace que existan ciertas reglas a la hora de manejar estados con hooks:

- **No llamar hooks dentro de condicionales, ciclos o funciones anidadas**
- **Llama a los hooks solo dentro de los componentes de React**
- **Llama a los hooks en el principio de la función del componente**

Lo anterior se propone para evitar re-renders infinitos en tus aplicaciones. Dale un vistazo a este gist sobre lo que no debes hacer al invocar un hook.

...

// mal - el hook esta dentro de un condicinoal

```
function MiComponente(props) {
  if(algunaCondicion) {
    let algunEstado = useState();
  }
  return (
```



```
<div>...</div>
)
}
```

// muy mal - el hook esta dentro del JSX

```
function MiComponente(props) {
  return (
    <div>
      {let algunEstado = useState()}
    </div>
  )
}
```

// muy muy mal - el hook esta dentro de un proceso iterativo

```
function MiComponente(props) {
  let lista = [1,2,3];
  lista.map(() => {
    let algunEstado = useState();
  })
  return (
    <div>
      ...
    </div>
  )
}
```



```
}  
  
// bien  
  
function MiComponente(props) {  
  // declara los hooks de primero en tus componentes  
  
  let algunEstado = useState();  
  
  return (  
    <div>...</div>  
  )  
}  
...  
...
```

En resumen

Una aplicación dinámica debe tener la capacidad de ofrecer interacciones y reacciones. Para que una aplicación sea dinámica, es necesario que sus componentes tengan estados, y que puedan transicionar entre ellos. El estado es el valor de uno o varios componentes en un momento determinado.

React nos permite manejar estados dentro de nuestros componentes con el hook `useState()`. El hook, que no es más que una función, permite setear el valor inicial de un estado, y a la vez nos retorna el valor del estado actual, junto a la función de actualización del estado.

Con estas dos piezas, podemos referenciar el valor actual del estado en nuestro JSX, así como invocar la función actualizadora dentro de la ocurrencia de eventos.



En la próxima meeting seguiremos explorando el concepto de estados en componentes, implementando otros tipos de valores. Además, el concepto de estados será tema de estudio desde hoy, hasta que finalices el programa.



Extra: renderizado condicional en componentes

Esta técnica consiste en aprovechar la lógica condicional de Javascript para mostrar y ocultar componentes, o partes gráficas de nuestra interfaz según una condición. Observa la sintaxis en el siguiente video:


[Renderizado condicional en componentes](#)


¡Prepárate para la próxima meeting!

Profundiza

-  Aprende un poco más sobre los hooks en [la documentación oficial](#) de React.
-  Conoce más sobre las reglas de hooks, [aquí](#).

Challenge

 1. Queremos que te familiarices con el patrón de actualización de estados con el hook 'useState()', por eso, te invitamos a replicar el ejercicio que viste en la toolbox. Parte de [este sandbox](#) para eso.

 2. Desarrolla una aplicación que te permita mostrar una lista de personas asistentes a un evento. Crea el componente `<Asistente />` el cual deberá tomar una prop representando el nombre, por ejemplo:

```
<Asistente nombre="Juan" />
```

Este componente debe mostrar por pantalla el nombre del asistente junto a un checkbox, cuyo estado debe estar controlado por React. Implementa varios asistentes. Mira [este gif](#) para tener una idea del resultado final.





3. Partiendo del [ejemplo de renderizado condicional](#) presentado en la toolbox, agrega los cambios necesarios para mostrar/ocultar el corazón al hacer click sobre el mismo. Agrega un estado que se alterne entre true y false, y vincúlalo al renderizado condicional del JSX.

