DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Front End III

setState() en acción

Volvamos a pensar el ejemplo de un post que dentro tenga un contador de likes. Ahora con el **useState()** sabemos que la cantidad de likes de nuestro post se preservará incluso luego de un re-renderizado. Vamos ahora a ver cómo actualizaremos ese estado.

Recordemos cómo se veía nuestro componente:

```
//Esto no funcionaria

const Post = (props) => {
  let likes = 0
  return (
    <div>
        <h1>{likes}</h1>
    </div>
  )}
```

Ahora cambiemos la variable likes por un estado interno dentro del componente:

```
const Post = (props) => {
  const [likes, setLikes] = useState(0);
```

```
  return (
    <div>
        <h1>{likes}</h1>
    </div>
  )}
```

Notemos que el nombre que pongamos a nuestro estado es indiferente dado que estamos haciendo un **destructuring**, donde lo que importa es la posición en el array y no el nombre en sí. Siempre la primera posición será el valor del estado (en este caso likes), y la segunda la función que nos permitirá **mutar** nuestro state. Generalmente, nombramos la función de actualizadora con la palabra **set + nombre de nuestro estado**. En esta ocasión, nombramos a nuestro estado como **likes**, por lo que nuestra función actualizadora será **setLikes**.

Veamos como quedaría ahora nuestro componente utilizando nuestra función para actualizar la cantidad de likes:

```
const Post = (props) => {
  const [likes, setLikes] = useState(0);
  return (
    <div>
        <h1>{likes}</h1>
        <button onClick={ ()=> setLikes(likes+1) }>
          ♥
        </button>
    </div>
  )}
```

Si bien pareciera que simplemente estamos agregando una unidad al valor likes, en realidad mediante el **setLikes** estamos creando un estado completamente nuevo.

El siguiente ejemplo muestra su implementación si tuviéramos más de un valor almacenado en el mismo estado:

```
const Post = (props) => {
  const [state, setState] = useState({
    likes: 0,
    vistas: 0
  });

  return (
    <div>
     //Accedemos al los likes y vistas por su key
        <h1>{state.likes}</h1>
        <h1>{state.vistas}</h1>

     //Con prev accedemos al valor "previo al cambio" del estado
    <button onClick={ ()=> setState(prev => ({...prev, likes: likes+1}))
}>
          ♥
    </button>
 </div>
  )}
```

Ahora, en vez de almacenar un único valor, estamos guardando un objeto. Luego, para acceder a cada valor de nuestro estado utilizamos la misma sintaxis que utilizamos para

acceder a cualquier objeto (key/value). Y ahora, para actualizar nuestro estado, la cosa cambia un poco. Veamos a fondo la sintaxis:

```
setState(prev => ({...prev, likes: likes+1}))
```

Como veremos **prev** es un parámetro (propio de la función set) que nos da acceso al valor anterior que tenía el estado. Para actualizar el valor del state, ahora haremos una copia de todo lo que teníamos anteriormente (con el spread operator) y cambiamos solamente el atributo que deseemos modificar.

Cabe destacar que **prev** no es una palabra reservada del lenguaje, sino que simplemente es un parámetro y podemos nombrarlo a gusto.