Hooks

Que son?

- Permiten usar características de React sin tener que escribir clases
- Permiten desarrollar componentes React usando sólo funciones
- React kooks nos brinda una serie de beneficios:
  - Reusar código se convierte en trivial. Con las clases no lo era
  - Componentes más simples
  - Beneficios a nivel de optimización debido al funcionamiento interno de React (AoT compilation, minification, ...)

useState

```
function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useEffect (Render)

```
import { useEffect } from 'react';

useEffect(() => {
  // El código se ejecuta en cada re-render
});
```

useEffect (Mount)

```
import { useEffect } from 'react';

useEffect(() => {
  // El código sólo se ejecuta en el primer render
}, []);
```

useEffect (Unmount)

```
import { useEffect } from 'react';

useEffect(() => {
  // Lógica del effect

  return () => {
    // Lógica de cleanup
  };
}, []);
```

useEffect (Property change)

```
import { useEffect } from 'react';

function SomeComponent(props) {
    useEffect(() => {
      // Lógica se ejecuta cada vez que cambia una variable
    }, [arrOfDependency, values, props.id]); // array de dependencias a checkear en cada render
}
```

useReducer

```
import { useReducer } from 'react';

function reducer(currentState, action) {
  switch(action.type) {
     // Gestión de cada una de las acciones utilizando action.payload
  }
}

function SomeComponent() {
  const [state, dispatch] = useReducer(reducer, initialState);

  dispatch({ type: 'ADD', payload: data }); // { type: 'ADD', payload: data } se pasa al reducer como la `action`
}
```

Creando hooks

```
import { useEffect } from 'react';

function useUser(userId) {
  let [user, setUser] = useState(null);

  useEffect(() => {
    fetch(`/api/user/${userId}`)
        .then(data => data.toJSON())
        .then(data => setUser(data));
  }, [userId]);

  return user;
}

function SomeComponent(props) {
  const user = useUser(props.id);
}
```

useMemo

memoize de resultados de cálculos complejos

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

useCallback

memoize de funciones

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useRef

Obtención de una ref

```
const refContainer = useRef(initialValue);

...

function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` apunta al elemento de entrada de texto montado
    inputEl.current.focus();
  };
  return (
    <React.Fragment>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </React.Fragment>
  );
}
```

Custom hooks

Un custom hook es una función Javascript cuyo nombre comienza con "use" y que usa otros hooks

```
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

Reglas

- Sólo invocarlos en el top-level del componente. No usarlos bajo condiciones, bucles o funciones anidadas
  - Aseguramos que la invocación se realizará siempre en el mismo orden en el render
  - Aseguramos que el estado se preservará de forma correcta entre renders
- No utilizarlos desde funciones Javascript normales
  - Desde componentes React
  - En custom hooks

Estas reglas aplican tanto a componentes como custom hooks

Tips

- https://dev.to/trentyang/replace-lifecycle-with-hooks-in-react-3d4n