React Hooks CheatSheet

by Ndeye Fatou Diop

What Are Hooks?

- Hooks are special functions in React that allow you to use state and other React features without writing a class.
- Introduced in React 16.8, they enable functional components to manage state, side effects, context, and more.

Rules of hooks

- You should only call hooks from within the body of a functional component or a custom hook.
- Do not call hooks conditionally (e.g: inside loops, conditions, etc.)
- Always call hooks at the top level of the component to ensure they run in the same order every time the component renders.
- Every hook name starts with “use”

useEffect hook

- useEffect lets you perform side effects (e.g., data fetching, subscriptions, manual DOM manipulation) in functional components.

useContext hook

- useContext allows components to subscribe to React context.

```
function ThemedButton()
{
  const theme =
useContext(ThemeContext
);
  return (<button
style={{ background:
theme.background }}>
    Click Me
  </button>);
}
```

useState hook

- useState is used to add state to functional components.
- Returns an array with the current state and a function to update it.

```
const [value, setValue] = useState(initialValue);
```

Custom Hooks

- Create your own hooks to reuse stateful logic across multiple components.
- Custom hooks follow the same rules as React hooks (start with use).

```
function useCurrentTime() {
  const [time, setTime] =
useState(new Date());

  useEffect(() => {
    const intervalId =
setInterval(() => {
      setTime(new Date());
    }, 1_000);
    return () =>
clearInterval(intervalId);
  }, []);

  return time;
}
```

useReducer hook

- useReducer is an alternative to useState for managing complex state logic.
- Ideal for scenarios where the state is complex and state updates depend on the previous state.

```
import { useReducer } from 'react';
import { reducer } from './reducer';

function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
      <span>{state.count}</span>
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
    </>
  );
}
```

useRef hook

- useRef returns a mutable ref object that persists across re-renders.
- Can be used to access DOM elements or store values that don’t need a re-render when changed.

```
const intervalIdRef = useRef();
const intervalId =
intervalIdRef.current;
  // Rest of logic…

  const stopTimer = () => {
    intervalId &&
clearInterval(intervalId);
  };
```