6.2 | Common hooks in react

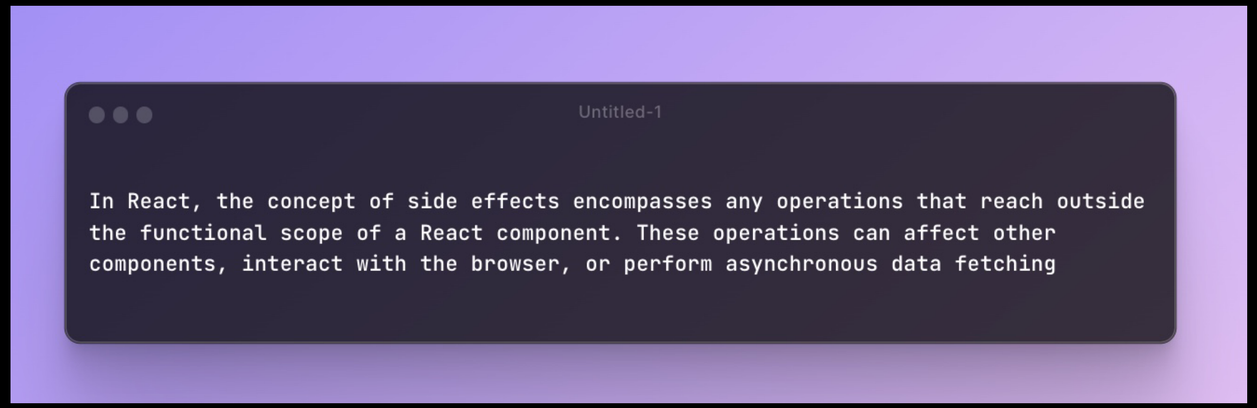
useEffect, useCallback, useMemo, custom hooks
Prop drilling

Two jargons before we start

1. Side effects
2. Hooks

Two jargons before we start

1. Side effects
2. Hooks



Two jargons before we start

1. Side effects
2. Hooks

Hooks are a feature introduced in React 16.8 that allow you to use state and other React features without writing a class. They enable functional components to have access to stateful logic and lifecycle features, which were previously only possible in class components. This has led to a more concise and readable way of writing components in React.

Two jargons before we start

1. Side effects
2. Hooks

Some common hooks are

1. useState
2. useEffect
3. useCallback
4. useMemo
5. useRef
6. useContext

useState

Let’s you describe the state of your app
Whenever state updates, it triggers a re-render which finally results in a DOM update

App.jsx > ...

```jsx
import { useState } from "react";

function App() {
  const [count, setCount] = useState(0)

  return <div>
    <button onClick={function() {
      setCount(count + 1);
    }}>Click me {count}</button>
  </div>
}

export default App;
```

useEffect

The `useEffect` hook is a feature in React, a popular JavaScript library for building user interfaces. It allows you to perform side effects in function components. Side effects are operations that can affect other components or can't be done during rendering, such as data fetching, subscriptions, or manually changing the DOM in React components.

The `useEffect` hook serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React class components, but unified into a single API.

Lets start with an example

You are a car racer that has to do a 100 laps across a stadium
You are allowed to take a pit stop from time to time.
Do you take the stop in b/w every lap? Or do you take a stop after every 10 laps lets say?

Lets start with an example

You will only make a pit stop
From time to time
(Lets say once every 20 laps)
even though you pass right in front of it
in every lap

Making a pit stop is a side effect

useEffect

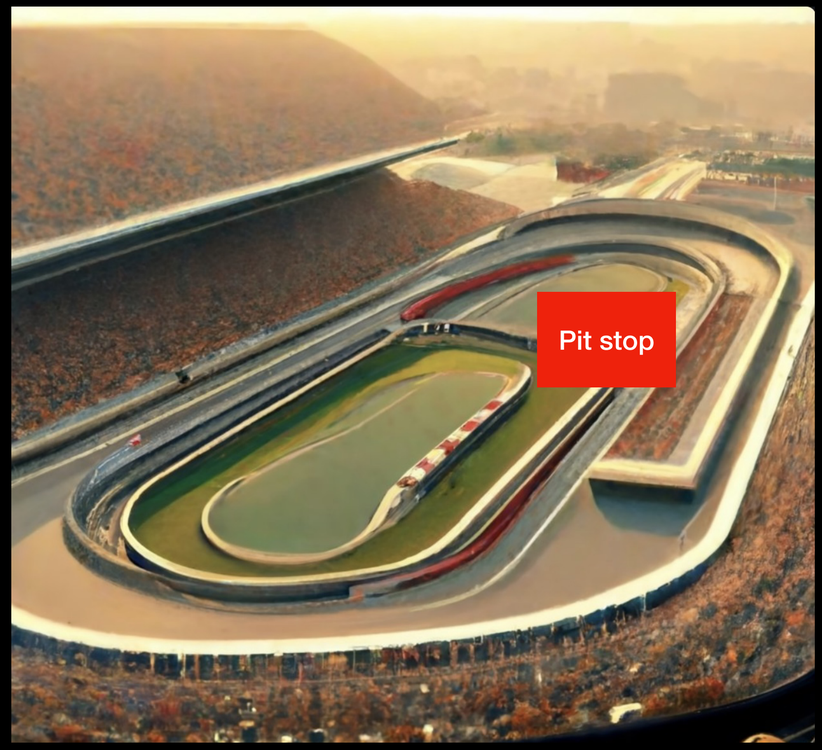
The `useEffect` hook is a feature in React, a popular JavaScript library for building user interfaces. It allows you to perform side effects in function components. Side effects are operations that can affect other components or can't be done during rendering, such as data fetching, subscriptions, or manually changing the DOM in React components.

The `useEffect` hook serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React class components, but unified into a single API.

useEffect

src > App.jsx > ...

```jsx
import { useState } from "react";
import { useEffect } from "react";

function App() {
  const [todos, setTodos] = useState([])

  useEffect(() => {
    fetch("https://sum-server.100xdevs.com/todos")
      .then(async function(res) {
        const json = await res.json();
        setTodos(json.todos);
      })
  }, [])

  return <div>
    {todos.map(todo => <Todo key={todo.id} title={todo.title} description={todo.description} />)}
  </div>
}

```



https://gist.github.com/hkirat/d3a74a1f4eff2c92f7a3c0da0c47d9c3

useEffect

```
useEffect(() => {
  fetch("https://sum-server.100xdevs.com/todos")
    .then(async function(res) {
      const json = await res.json();
      setTodos(json.todos);
    })
}, [])
```

What should happen?

useEffect

```
useEffect(() => {
  fetch("https://sum-server.100xdevs.com/todos")
    .then(async function(res) {
      const json = await res.json();
      setTodos(json.todos);
    })
}, [])
```

When should it happen?

useEffect

1. Tyre burst
2. Tyre pressure is up
3. 10 laps have passed
4. Engine is making a noise
5. Want to change the car

```
useEffect(() => {
  fetch("https://sum-server.100xdevs.com/todos")
    .then(async function(res) {
      const json = await res.json();
      setTodos(json.todos);
    })
}, [])
```

Dependency array

When should the callback fn run

useEffect

Write a component that takes a todo id as an input
And fetches the data for that todo from the given endpoint
And then renders it

How would the dependency array change?

https://sum-server.100xdevs.com/todo?id=1

src > App.jsx > Todo

```jsx
import { useState } from "react";
import { useEffect } from "react";

function App() {
  return <div>
    <Todo id={1} />
  </div>
}

function Todo({id}) {
  const [todo, setTodo] = useState({});

  // your effect here

  return <div>
    <h1>
      {todo.title}
    </h1>
    <h4>
      {todo.description}
    </h4>
  </div>
}

export default App;
```

Solution
https://gist.github.com/hkirat/178abff7b3bc80af5878be7b9a3b7d69

useEffect

useEffect

```
useEffect(() => {
  fetch("https://sum-server.100xdevs.com/todo?id=" + id)
    .then(async function(res) {
      const json = await res.json();
      setTodo (json.todo);
    })
}, [id])
```

useEffect

useEffect, useMemo, useCallback, custom hooks
Prop drilling

useMemo

Before we start , lets understand what memoization means

It’s a mildly DSA concept

It means remembering some output given an input and not computing it again

useMemo

Lets say you are the driver
and you want to check how much petrol is left
Would u do that in every lap?
Or would u do that every 10 laps?
Or every 20 minutes?

useMemo

Lets say you are the driver
and you want to check how much petrol is left
Would u do that in every lap?
Or would u do that every 10 laps?
Or every 20 minutes?

useMemo

If I ask you to create an app that does two things -

1. Increases a counter by 1
2. Lets user put a value in an input box (n) and you need to show sum from 1-n

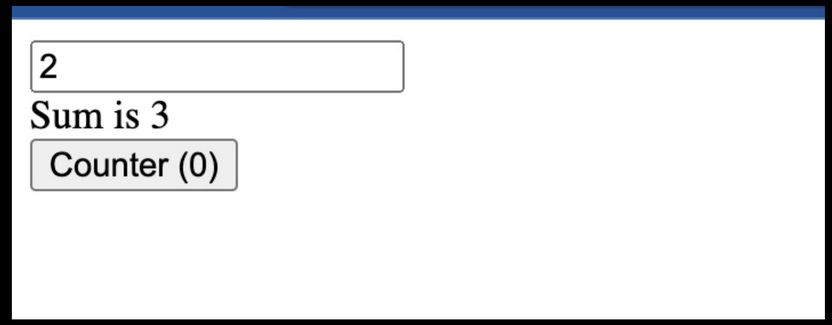
One restriction - everything needs to be inside App

useMemo

Ugly solution

src > App.jsx > App

```jsx
import { useState } from "react";

function App() {
  const [counter, setCounter] = useState(0);
  const [inputValue, setInputValue] = useState(1);


  let count = 0;
  for (let i = 1; i <= inputValue; i++) {
    count = count + i;
  }

  return <div>
    <input onChange={function(e) {
      setInputValue(e.target.value);
    }} placeholder={"Find sum from 1 to n"}></input>
    <br />
    Sum from 1 to {inputValue} is {count}
    <br />
    <button onClick={() => {
      setCounter(counter + 1);
    }}>Counter ({counter})</button>
  </div>
}

export default App;

```



https://gist.github.com/hkirat/30d07fc535c83ab89b7b9c58b11eff4f

useMemo

Better solution - memoize the value across re-renders, only recalculate it if inputVal changes

useEffect, useMemo, useCallback, custom hooks
Prop drilling

useCallback

`useCallback` is a hook in React, a popular JavaScript library for building user interfaces. It is used to memoize functions, which can help in optimizing the performance of your application, especially in cases involving child components that rely on reference equality to prevent unnecessary renders.

useCallback

What is the problem in this code?

src > App.jsx > ...

```jsx
import { memo, useState } from "react";

function App() {
  const [count, setCount] = useState(0)

  function onClick() {
    console.log("child clicked")
  }

  return <div>
    <Child onClick={onClick} />
    <button onClick={() => {
      setCount(count + 1);
    }}>Click me {count}</button>
  </div>
}

const Child = memo(({onClick}) => {
  console.log("child render")

  return <div>
    <button onClick={onClick}>Button clicked</button>
  </div>
})

export default App;

```

https://gist.github.com/hkirat/34c2ddf04284945f7bf0f17cbe8347f6

useCallback

Solution

src > App.jsx > App > onClick

```jsx
import { memo, useCallback, useState } from "react";

function App() {
  const [count, setCount] = useState(0)

  const onClick = useCallback(() => {
    console.log("child clicked")
  }, [])

  return <div>
    <Child onClick={onClick} />
    <button onClick={() => {
      setCount(count + 1);
    }}>Click me {count}</button>
  </div>
}

const Child = memo(({onClick}) => {
  console.log("child render")

  return <div>
    <button onClick={onClick}>Button clicked</button>
  </div>
})

export default App;
```

useEffect, useMemo, useCallback, custom hooks
Prop drilling

Custom hooks

Just like useState, useEffect, you can write your own hooks

Only condition is - It should start with a use (naming convention)

Custom hooks

```jsx
import { useEffect } from "react";

function App() {
  const [todos, setTodos] = useState([])

  useEffect(() => {
    fetch("https://sum-server.100xdevs.com/todos")
      .then(async function(res) {
        const json = await res.json();
        setTodos(json.todos);
      })
  }, [todos])

  return <div>
    {todos.map(todo => <Todo key={todo.id} title={todo.title} description={todo.description} />)}
  </div>
}

function Todo({title, description}) {
  return <div>
    <h1>
      {title}
    </h1>
    <h4>
      {description}
    </h4>
  </div>
}

export default App;
```

```jsx
function App() {
  const todos = useTodos();

  return <div>
    {todos.map(todo => <Todo key={todo.id} title={todo.title} description={todo.description} />)}
  </div>
}

function Todo({title, description}) {
  return <div>
    <h1>
      {title}
    </h1>
    <h4>
      {description}
    </h4>
  </div>
}

export default App;
```

useEffect, useMemo, useCallback, custom hooks

Prop drilling

Prop drilling