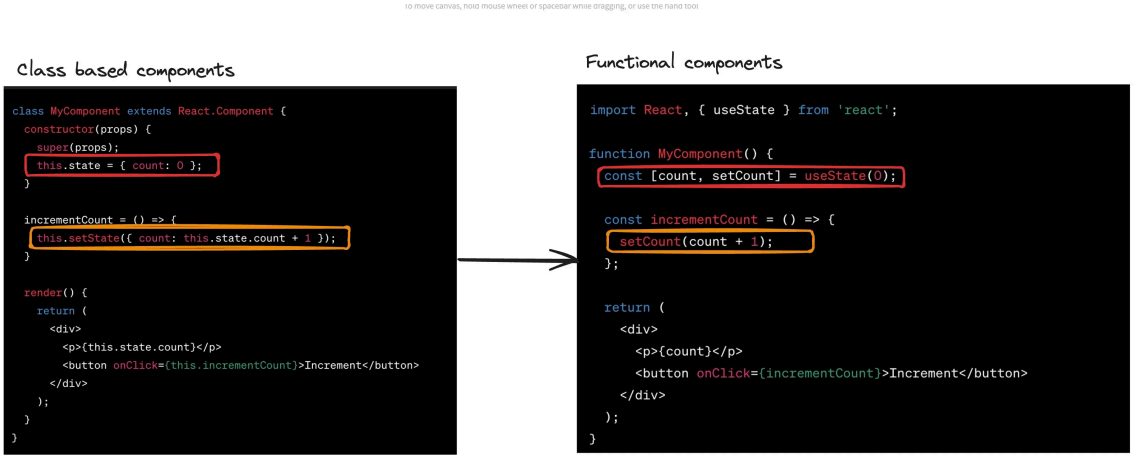
1 - What are hooks

What are hooks

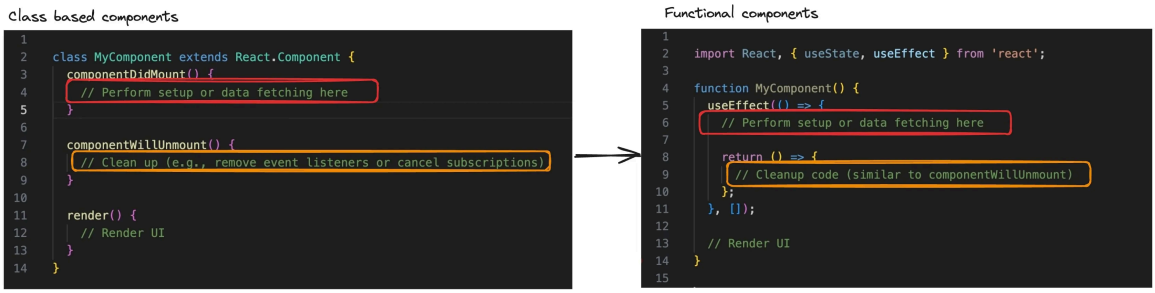
Hooks are a feature introduced in React 16.8 that allow you to use state and other React features without writing a class. They are functions that let you "hook into" React state and lifecycle features from function components.

State



- ▶ Functional
- ▶ Class Based

Lifecycle events



- Functional
- Class based
- Functional solution

Until now we're seen some commonly used hooks in React-

1. useState
2. useEffect
3. useMemo
4. useCallback

These hooks are provided to you by the React library.

2 - What are custom hooks

Hooks that you create yourself, so other people can use them are called custom hooks.

A custom hook is effectively a function, but with the following properties -

1. Uses another hook internally (useState, useEffect, another custom hook)
2. Starts with use

A few good examples of this can be

1. Data fetching hooks
2. Browser functionality related hooks - **useOnlineStatus , useWindowSize, useMousePosition**
3. Performance/Timer based - **useInterval, useDebounce**

3 - Data fetching hooks

Data fetching hooks can be used to encapsulate all the logic to fetch the data from your backend

For example, look at the following code-

```
import { useEffect, useState } from 'react'
import axios from 'axios'

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

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
      })
  }, [])

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}

export default App
```

Step 1 - Converting the **data fetching** bit to a custom hook

```
import { useEffect, useState } from 'react'
import axios from 'axios'

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

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
      })
  }, [])

  return todos;
}

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

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}
```

```
export default App
```

Step 2 - Cleaning the hook to include a `loading` parameter

What if you want to show a loader when the data is not yet fetched from the backend?

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos() {
  const [loading, setLoading] = useState(true);
  const [todos, setTodos] = useState([])

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
        setLoading(false);
      })
  }, [])

  return {
    todos: todos,
    loading: loading
  };
}

function App() {
  const { todos, loading } = useTodos();

  if (loading) {
    return <div>
      Loading...
    </div>
  }

  return (
```

```
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}

export default App
```

Step 3 - Auto refreshing hook

What if you want to keep polling the backend every n seconds?

n needs to be passed in as an input to the hook

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos(n) {
  const [loading, setLoading] = useState(true);
  const [todos, setTodos] = useState([])

  function getData() {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
        setLoading(false);
      })
  }

  useEffect(() => {
    setInterval(() => {
      getData();
    }, n * 1000)
    getData();
```

```
  }, [n])

  return {
    todos: todos,
    loading: loading
  };
}

function App() {
  const { todos, loading } = useTodos(5);

  if (loading) {
    return <div>
      Loading...
    </div>
  }

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}

export default App
```

▶ Final solution

swr - React Hooks for Data Fetching

swr is a popular React library that creates a lot of these hooks for you, and you can use it directly.

For example -

```
import useSWR from 'swr'

// const fetcher = (url) => fetch(url).then((res) => res.json());
const fetcher = async function(url) {
  const data = await fetch(url);
  const json = await data.json();
  return json;
};

function Profile() {
  const { data, error, isLoading } = useSWR('https://sum-server.100x

  if (error) return <div>failed to load</div>
  if (isLoading) return <div>loading...</div>
  return <div>hello, you have {data.todos.length} todos!</div>
}
```

https://swr.vercel.app/

4 - Browser functionality related hooks

1. useIsOnline hook

Create a hook that returns true or false based on weather the user is currently online

You are given that -

1. window.navigator.onLine returns `true` or `false` based on weather the user is online
2. You can attach the following event listeners to listen to weather the user is online or not

```
window.addEventListener('online', () => console.log('Became onlin
window.addEventListener('offline', () => console.log('Became offlin
```

▶ Solution

2. useMousePointer hook

Create a hook that returns you the current mouse pointer position.

The final react app that uses it looks like this

You are given that

```
window.addEventListener('mousemove', handleMouseMove);
```

will trigger the `handleMouseMove` function anytime the mouse pointer is moved.

▶ Solution

5 - Performance/Timer based

1. useInterval

Create a hook that runs a certain callback function every n seconds.

You have to implement `useInterval` which is being used in the code below -

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

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

 useInterval(() => {
  setCount(c => c + 1);
 }, 1000)

 return (
  <>
   Timer is at {count}
  </>
 )
}

export default App
```

Final app should look like this

▶ Solution

2. useDebounce

Create a hook that debounces a value given

1. The value that needs to be debounced
2. The interval at which the value should be debounced.

```
import React, { useState } from 'react';
import useDebounce from './useDebounce';

const SearchBar = () => {
  const [inputValue, setInputValue] = useState('');
  const debouncedValue = useDebounce(inputValue, 500); // 500

  // Use the debouncedValue in your component logic, e.g., trigger

  return (
    <input
      type="text"
      value={inputValue}
      onChange={(e) => setInputValue(e.target.value)}
      placeholder="Search..."
    />
  );
};
```

```
export default SearchBar;
```

▶ Solution