Custom Hooks

Download Demo Code

Goals

- Understand when it's best to create custom hooks
- Create a few helpful custom hooks

What are custom hooks

Custom Hooks

- A JavaScript function that typically uses built in hooks
- The function name of a custom hook should start with use (useToggleState)
- Custom hooks can be reusable across different components

Making your own hooks

- You can make hooks to
  - abstract logic
  - handle repeated tasks
  - generate your own JSX

Try to resist adding abstraction too early. Now that function components can do more, it's likely that the average function component in your codebase will become longer. This is normal — don't feel like you have to immediately split it into Hooks. But we also encourage you to start spotting cases where a custom Hook could hide complex logic behind a simple interface, or help untangle a messy component.

Creating custom hooks

Making a simple toggle hook

- One of the most common things you'll do in applications is toggle some state!
- This could be hiding or showing something, turning something on or off, or moving something to one place or another.
- We could repeat this logic over and over again, or we could use a hook that does it for us!

useToggleState

demo/hooks-starter/src/hooks/useToggleState.js

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

function useToggle(initialVal = false) {
  // call useState, "reserve piece of state"
  const [value, setValue] = useState(initialVal);
  const toggle = () => {
    setValue(oldValue => !oldValue);
  };

  // return piece of state AND a function to toggle it
  return [value, toggle];
}

export default useToggle;
```

- Note we return the piece of state and the toggle function, not ***setValue***!

A more complex hook

- Very commonly you'll find yourself fetching data when a component mounts
- Remember, our hooks can return anything! So we can make hooks that return JSX or return an object or array with functionality we'd like to re-use.
- You could re-write that loading, fetching and error handling logic every time, or we could make a hook!

Our useFetch hook

demo/hooks-starter/src/hooks/useFetch.js

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

const useFetch = (url, options = {}) => {
  const [response, setResponse] = useState(null);
  const [error, setError] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  // after the first render, fetch our data
  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await fetch(url, options);
        const json = await res.json();
        setResponse(json);
      } catch (error) {
        setError(error);
      }
      setIsLoading(false);
    };
    fetchData();
  }, []);

  return { response, error, isLoading };
};

export default useFetch;
```

Using our useFetch hook

demo/hooks-starter/src/DogDetail.js

```
import React from "react";
import useFetch from "./hooks/useFetch";

const DogDetail = () => {
  const data = useFetch("https://dog.ceo/api/breeds/image/random");
  if (data.isLoading) {
    return <div>Loading...</div>;
  }
  if (data.error) {
    return <div>Sorry, something went wrong :(</div>
  }
  const { status, message } = data.response;
  return (
    <div className="App">
      <div>
        <h3>{status}</h3>
        <div>
          <img src={message} alt="avatar" />
        </div>
      </div>
    </div>
  );
};

export default DogDetail;
```

When Should I Write My Own Hooks?

General Tips

You don't *have* to write custom hooks, but there are a number of situations where they can be useful.

- Same business logic inside of multiple components? Maybe a custom hook can help.
- Business logic cluttering up a single component? Maybe a custom hook, even if only for one component, can help with readability.
- Don't start by asking what custom hooks you need; lean on custom hooks to help you refactor your code.