# Custom Hooks in React

Have any of your components starting to get chunky with logic and functions? We’re going to talk about a strategy for dealing with that and make our logic and components more reusable.

We were asked to watch a video (<https://www.youtube.com/watch?v=Jl4q2cccwf0>) and article (<https://ziffur.com/article/composing_with_react_hooks>) to understand the basics first.

Takeaways from [the video](https://www.youtube.com/watch?v=Jl4q2cccwf0):

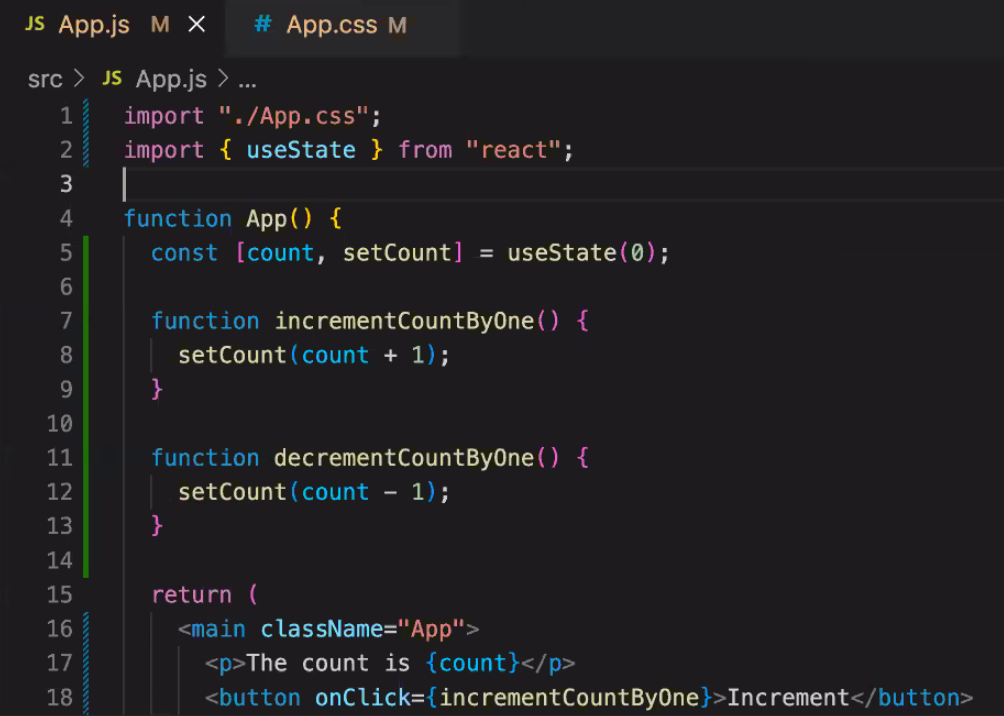
* Custom hooks allow you to reuse code in many components
  + This is particularly handy for big logic like useEffect
  + Or fetch requests
  + But can be used for any functionality!
* Create a file called useFetch.js in the src – you are basically creating a component for the custom hook (like we saw for useContext)
* Const useFetch = () => {}
* Custom hooks must start with the word “use”
* Can then copy and paste whatever code is already working inside it (make sure to import useState, useEffect etc and declare any state that you need)
  + Rename any state to be more generic (e.g. instead of Blogs, use Data)
* At the end of the whole file, remember to export it!
* Then need to return values (such as state) from the custom hook – at the bottom of the file outside of the useFetch function
  + Return as an object (e.g. return {data, isPending, error}
* Don’t hardcode URL – pass it in to the hook as an argument, then add it as a dependency array if needed
* Import in to app using const {data, isPending, Error} = useFetch(*arguments* (e.g. URL))
  + Can also specify data: blogs to import the data as blogs so you can still pass in props

Takeaway from [the article](https://ziffur.com/article/composing_with_react_hooks):

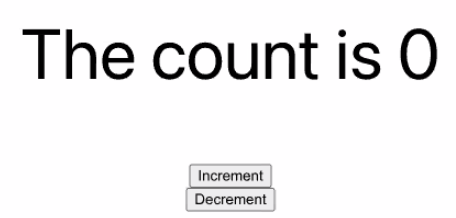
* Lots of duplication in Arhi’s example below
* Naming is really important –use verbs for functions, nouns for generic variables and yes/no questions (using the prefixes is/has/should) for Booleans
  + See here: https://github.com/kettanaito/naming-cheatsheet

## Arshi then demoed an example with a simple Counter app:

### Original code:

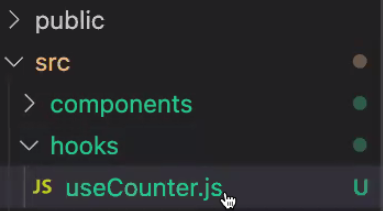


### In the browser:



* Notice there is a State for count, functions for increasing or decreasing the count.
* We are mixing concerns in this file – the logic for what it’s doing and the presentation of what is on the screen.
* All of the logic about the state and updating the state can be moved out to a custom hook.

One pattern for custom hooks is creating a folder called hooks in the src folder (separate to components). There are plenty of others, but whatever you choose – be consistent and remember to name the component semantically!



Plan:

Write a custom hook (function)

Export it (so that it’s available in other files)

Remember to import any other hooks being used (e.g. useState)

Function:

Don’t need to take in any input on this occasion but you may need to for others

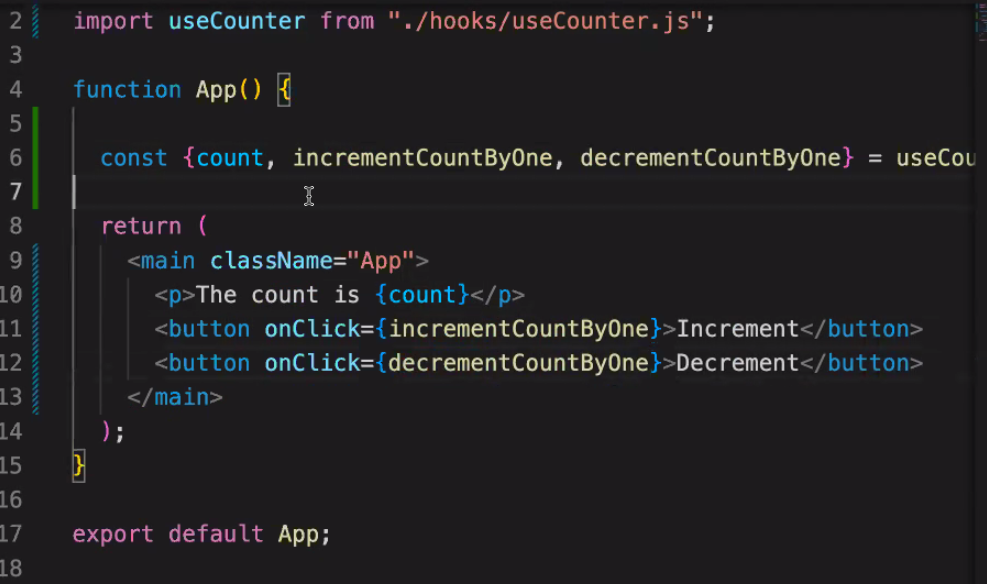
We need to give some things back (remember to return!)

### In useCounter.js:



### In app.js:

Import it in to the app (line 2) and de-structure the results when calling it (line 6)



## Arshi’s Golden Nuggets

* Custom hooks are very testable (as it will be a function that returns something)
  + One library for testing your hooks is React Hooks Testing Library ()
* Custom hooks can give you more control of what is being returned (e.g. you could make a custom error message instead of just error.message)
* Checkout [www.usehooks.com](http://www.usehooks.com) for examples of custom hooks
* We used a custom hook when implementing Auth0 (useAuth0):

