10 React Hooks you Should Have in Your Toolbox

The custom hooks you should probably have in your arsenal.

Aayush Jaiswal Following

Mar 29, 2019 · 7 min read

Photo by Joanna Kosinska on Unsplash

Hooks came and conquered React and shook the whole developer community. Scrolling down your twitter feed without noticing the word **hooks** is a rare occasion these days. And by this day, I hope you have accepted your fate and have adopted them in your codebase.

Have these thoughts ever crossed your mind when you are done with a code segment?

> *How can this code be optimized?* 🤔
>
> *Can the number of lines be reduced?*
>
> *I want to make the code much cleaner..* 🤩

Of course, they did and so came **Custom hooks** to our rescue. You probably know what are custom hooks and chances are you might have written them if you haven't do check out my previous article where I teach the art of writing them.

Writing Your Own Custom Hooks!

Let's extract your components logic into reusable functions.

blog.bitsrc.io

I prefer fewer keystrokes rather than overwriting the monotonous and repetitive code.

DRYness is what I crave😋 for and so should you.

It can be pretty gruesome writing custom hooks every time we encounter a different problem. Sure, it’s challenging and I know you love to encounter them, but there can be times that you are running short on time or have some deadlines that you have to keep up with.

But thanks to our awesome and nerdy React community that we have already been blessed with thousands of these hooks. It’s up to us to use them to their full potential and that is what this post is for, to make you aware of these wonderful hooks that every developer should have in his arsenal.

Without any further ado, let’s get into what you are here for.

> *Note: The order of the below list makes no sense, it’s random.*

> *Note: You can find and use all of the hooks in this collection:*

must-have-hooks-in-your-toolbox by aayush1408 · Bit

25 Javascript components. Examples: is-browsermacro , use-form-state , is-browser , constants , indexd , utils , index...

bit.dev

1. useArray hook

Array manipulation is what a dev goes through every weekday. Adding elements to an array or removing an element at a given index is a daily routine for us. `useArray` reduces this burden by providing us with various array manipulation methods. This hook is a part of the react-hanger package.

Installation:-

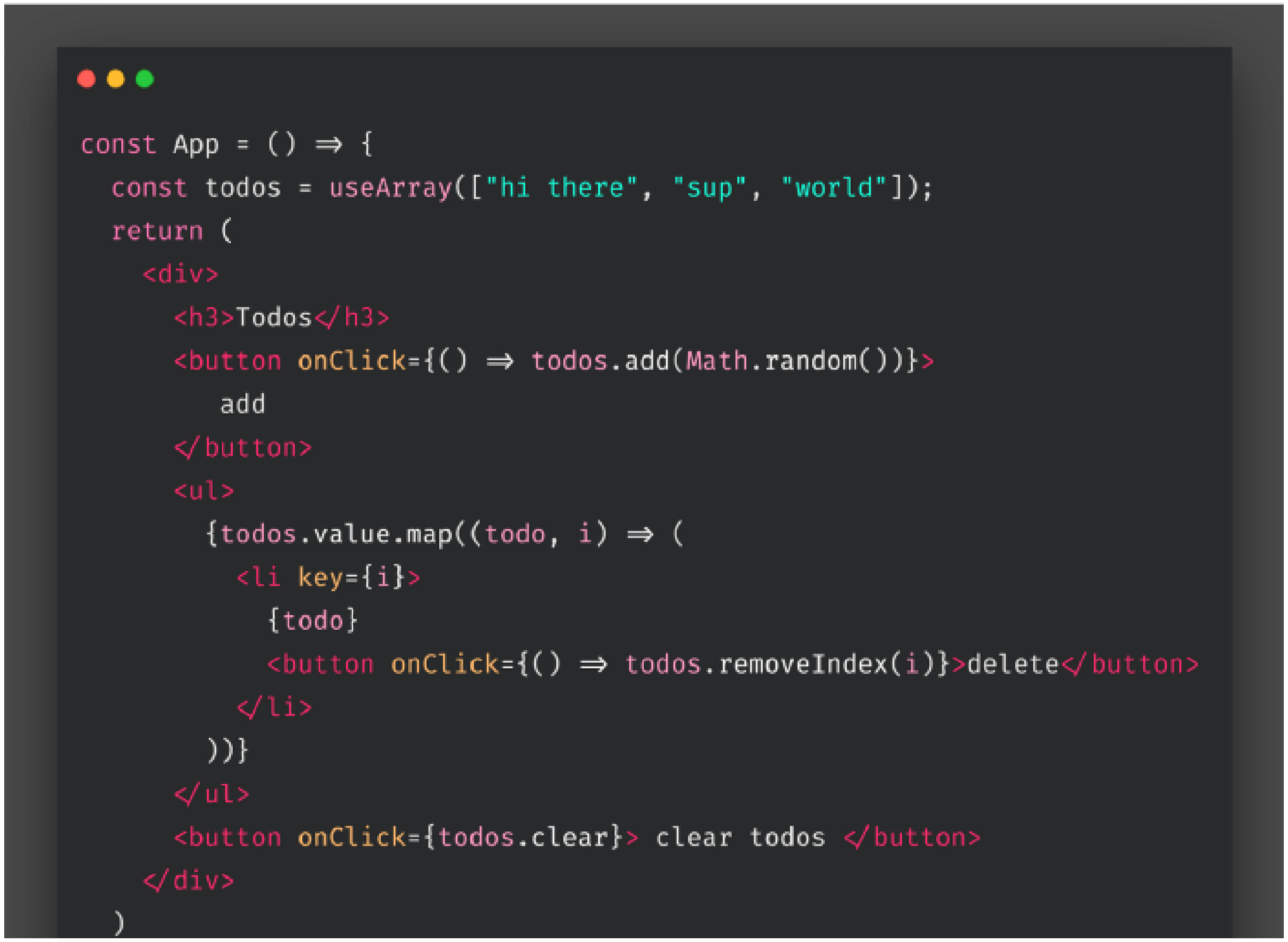
```
yarn add react-hanger
```

Including it in our file is super easy:

```
import {useArray} from 'react-hanger'
```

Usage:

Building a todo list was never this simple. We provide the array in the hook and get access to these methods and array in the todos object below.



```
const App = () => {
  const todos = useArray(["hi there", "sup", "world"]);
  return (
    <div>
      <h3>Todos</h3>
      <button onClick={() => todos.add(Math.random())}>
         add
      </button>
      <ul>
        {todos.value.map((todo, i) => (
          <li key={i}>
            {todo}
            <button onClick={() => todos.removeIndex(i)}>delete</button>
          </li>
        ))}
      </ul>
      <button onClick={todos.clear}> clear todos </button>
    </div>
  )
```

```
}
```

useArray hook

I have discussed its implementation in my previous article.

For a much detail look into the hook, you can check out the Bit component or the GitHub repo.

2. react-use-form-state hook

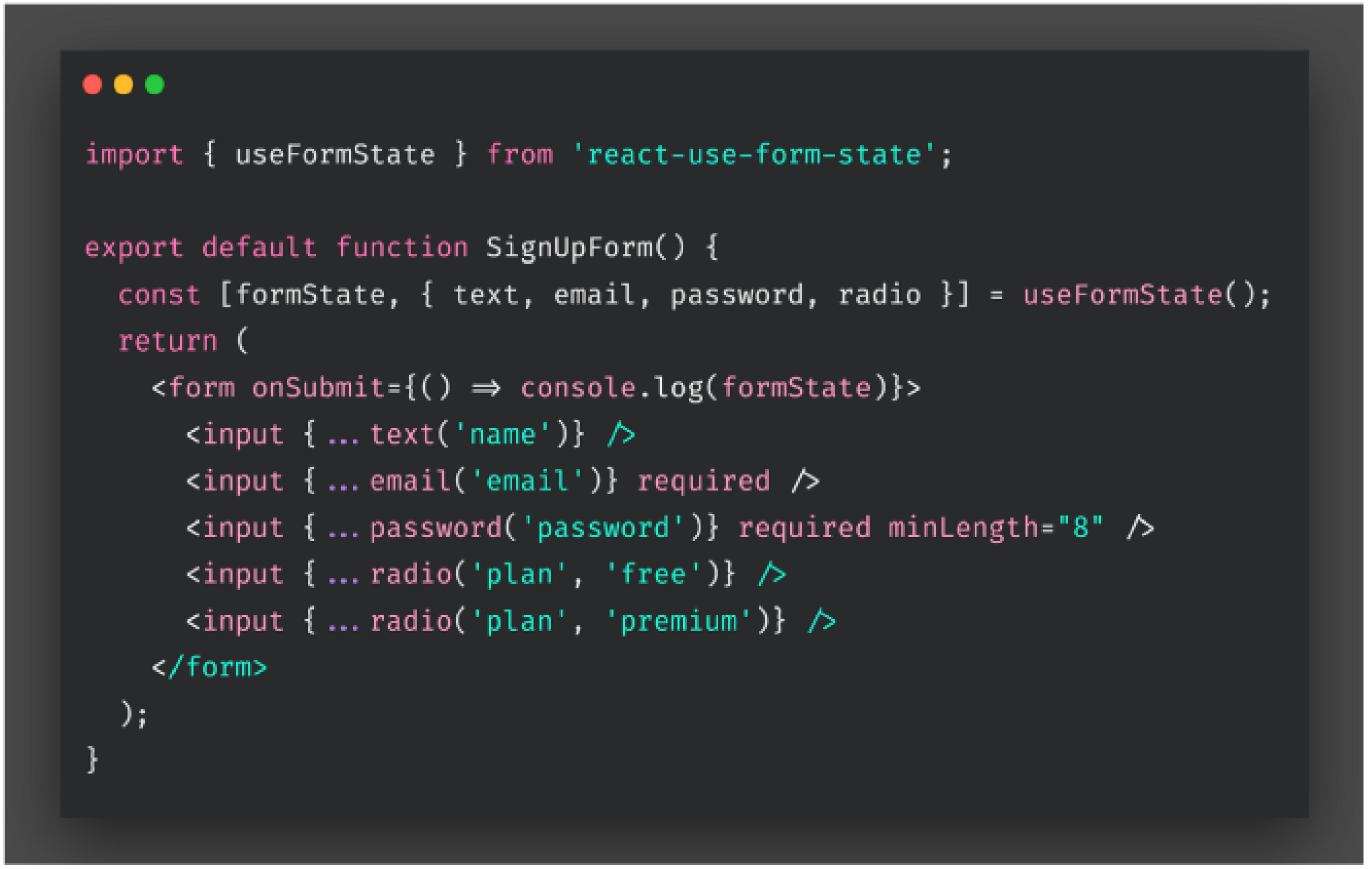
Forms are everywhere, even in the smallest of applications we have to encounter forms and manage their state. Managing form state in React can be a bit unwieldy sometimes.

`react-use-form-state` is a small React Hook that attempts to simplify managing form state, using the native form input elements you are familiar with.

Installation:

```
npm i react-use-form-state
```

Usage:



```
import { useFormState } from 'react-use-form-state';

export default function SignUpForm() {
  const [formState, { text, email, password, radio }] = useFormState();
  return (
    <form onSubmit={() => console.log(formState)}>
      <input {...text('name')} />
      <input {...email('email')} required />
      <input {...password('password')} required minLength="8" />
      <input {...radio('plan', 'free')} />
      <input {...radio('plan', 'premium')} />
    </form>
  );
}
```

useFormState hook

`formState` has a structure like this:

```
{
  "values": {
    "name": "Mary Poppins",
    "email": "mary@example.com",
    "password": "1234",
    "plan": "free",
  },
  "validity": {
    "name": true,
    "email": true,
    "password": false,
    "plan": true,
  },
  "touched": {
    "name": true,
    "email": true,
    "password": true,
    "plan": true,
  }
}
```

It's a much efficient way to keep track of the state of the form. There is more to it so please check out the docs.

For a much detail look into the hook, you can check out the Bit component or the GitHub repo.

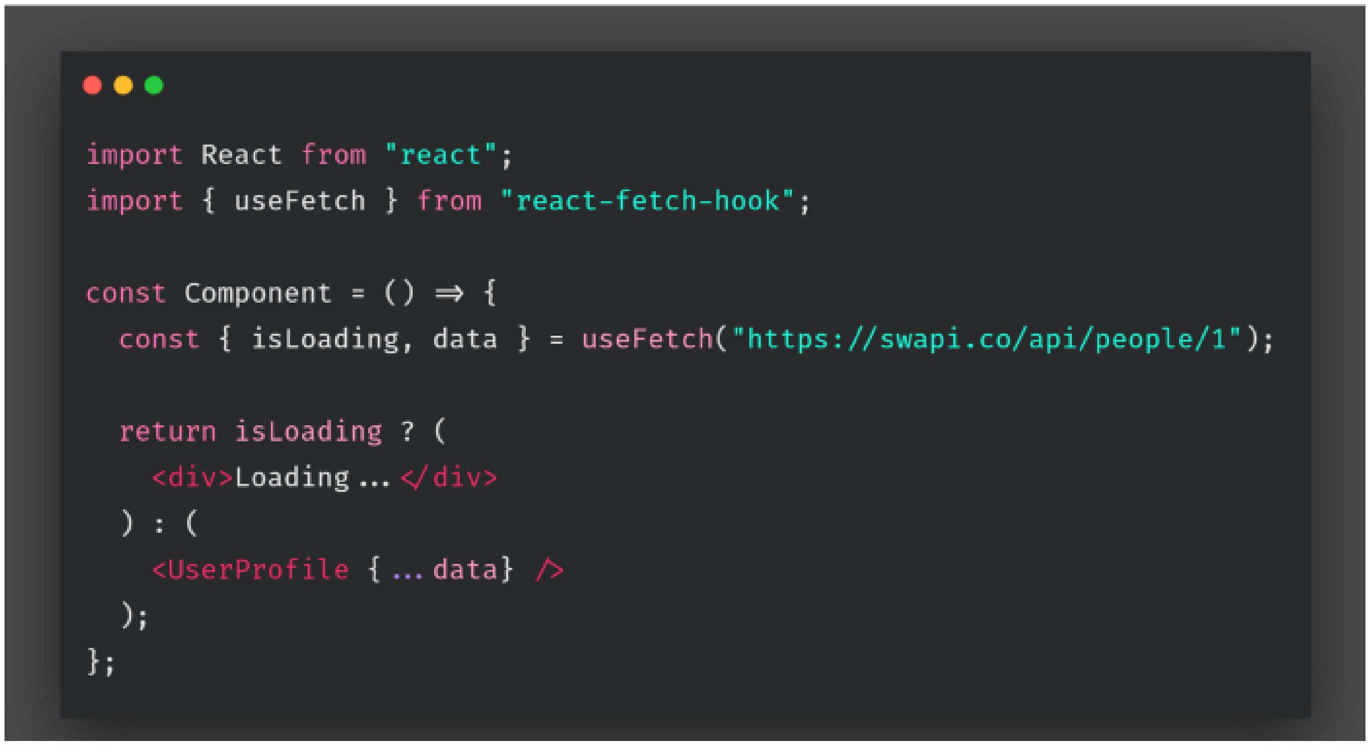
3. react-fetch-hook

Making ajax calls is like the most basic and most performed task for a frontend developer. And the React community is quick enough to create a hook for this purpose too.

Installation:

```
npm i react-fetch-hook
```

Usage:

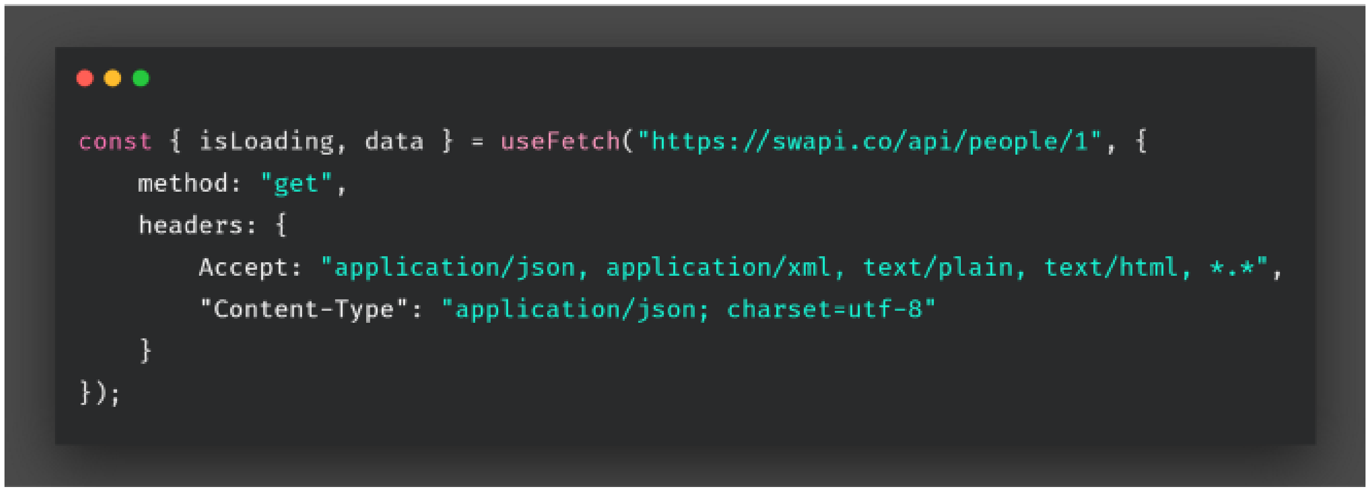


```
import React from "react";
import { useFetch } from "react-fetch-hook";

const Component = () => {
  const { isLoading, data } = useFetch("https://swapi.co/api/people/1");

  return isLoading ? (
    <div>Loading...</div>
  ) : (
    <UserProfile {...data} />
  );
};
```

useFetch hook

Not much to say, `useFetch` hook gets us the data and the `isLoading` state.

We can also provide the required options object to the hook.



```
const { isLoading, data } = useFetch("https://swapi.co/api/people/1", {
    method: "get",
    headers: {
        Accept: "application/json, application/xml, text/plain, text/html, *.*",
        "Content-Type": "application/json; charset=utf-8"
    }
});
```

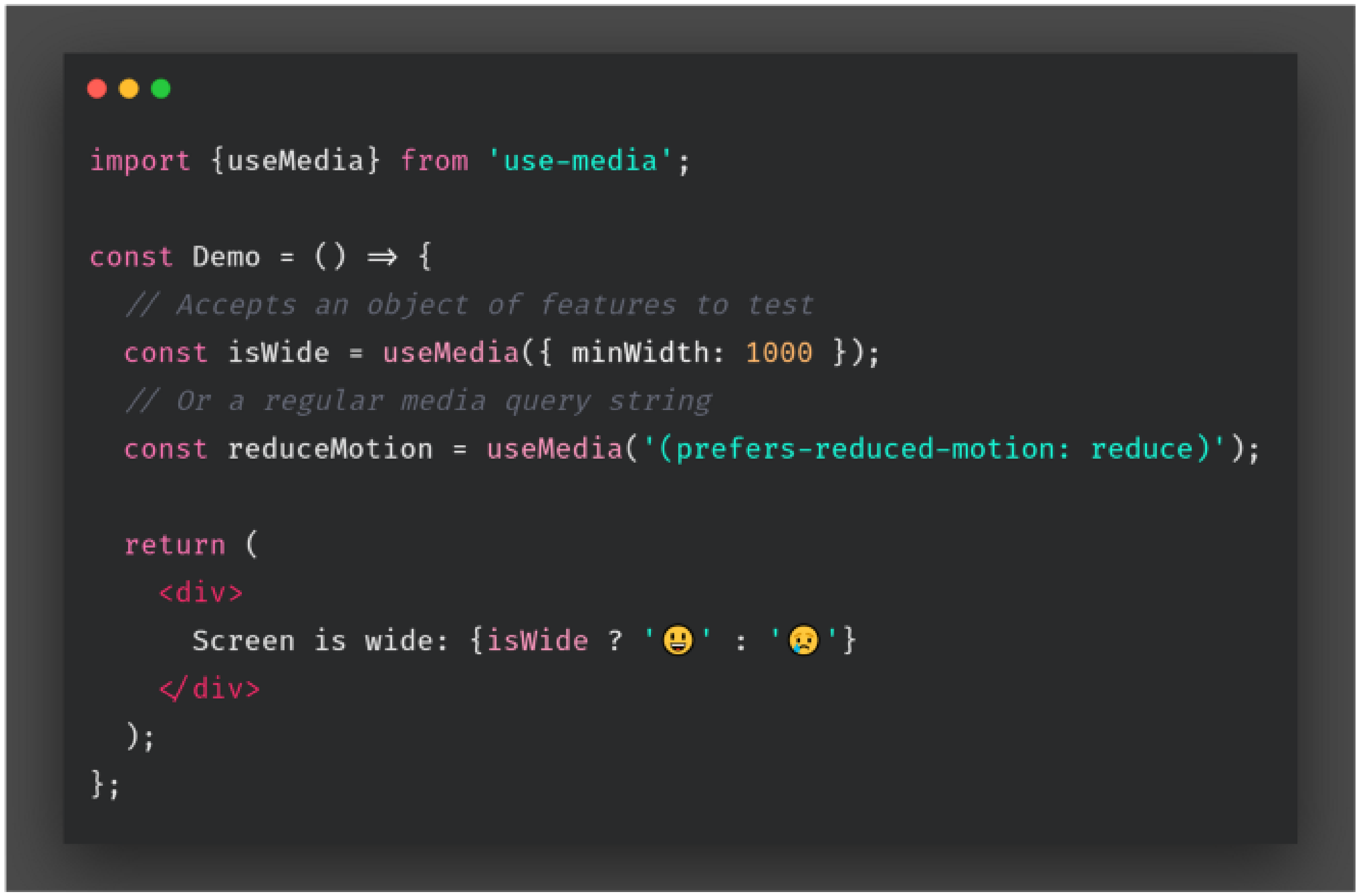
Options provided to the hook

For a much detailed look, you can check out the Bit component or the GitHub repo.

4. useMedia hook

`useMedia` is a React sensor hook that tracks the state of a CSS media query. We all know the importance of the media queries and how much important is responsiveness for any site.

Usage:



```
import {useMedia} from 'use-media';

const Demo = () => {
  // Accepts an object of features to test
  const isWide = useMedia({ minWidth: 1000 });
  // Or a regular media query string
  const reduceMotion = useMedia('(prefers-reduced-motion: reduce)');

  return (
    <div>
      Screen is wide: {isWide ? '😃' : '😢'}
    </div>
  );
};
```

useQuery hook

An object is provided to the hook, which returns a boolean response.

For a much detailed look, you can check out the Bit component or the GitHub repo.

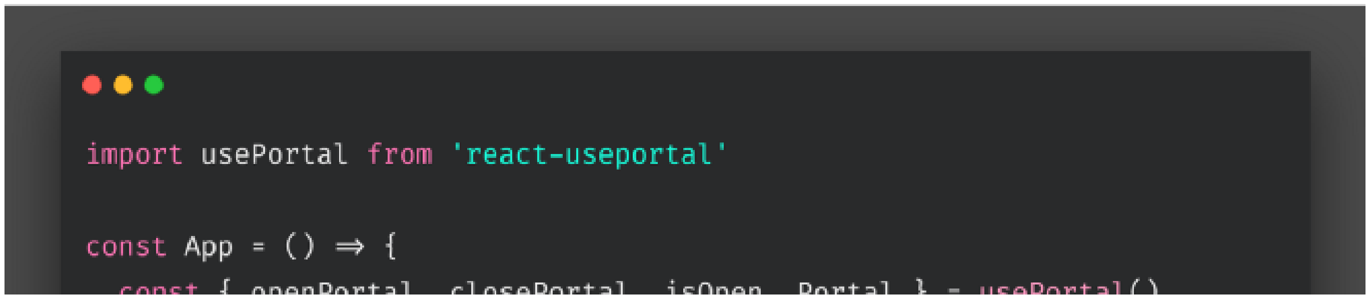
5. react-useportal hook

React Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component. And this hook helps us do that.

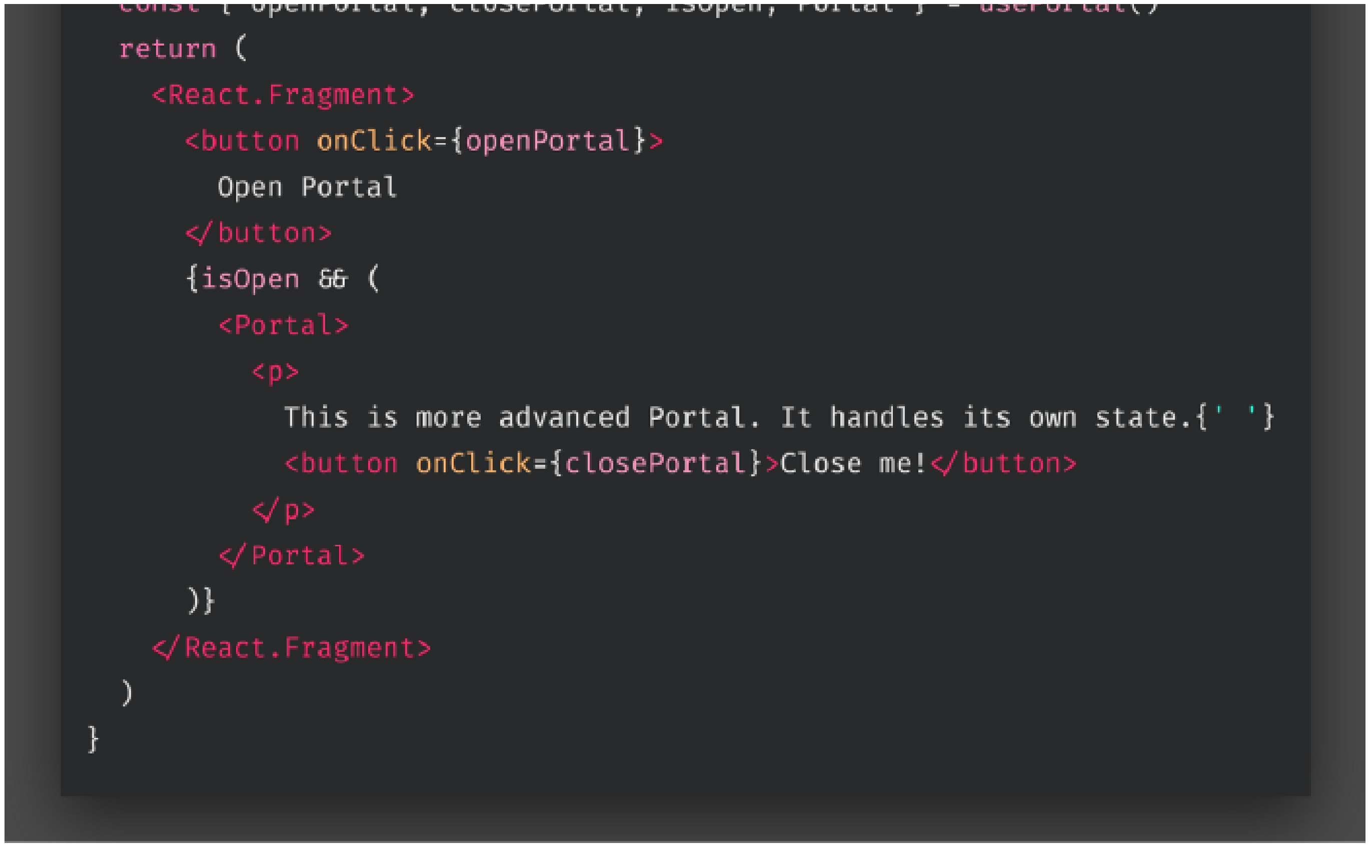
Installation:

```
yarn add react-useportal
```

Usage:



```
import usePortal from 'react-useportal'

const App = () ⇒ {
  const { openPortal, closePortal, isOpen, Portal } = usePortal()
```



```
  const { openPortal, closePortal, isOpen, Portal } = usePortal()
  return (
    <React.Fragment>
      <button onClick={openPortal}>
        Open Portal
      </button>
      {isOpen && (
        <Portal>
          <p>
            This is more advanced Portal. It handles its own state.{' '}
            <button onClick={closePortal}>Close me!</button>
          </p>
        </Portal>
      )}
    </React.Fragment>
  )
}
```

usePortal hook

This is pretty easy to use. This hook provides us with two methods `openPortal` and `closePortal` (one opens the portal and another close it), one boolean value `isOpen` to show the status of the portal and a component to wrap the content of the portal. There is a lot to this hook, so do check out the docs.

For a much detail look into the hook, you can check out the Bit component or the GitHub repo.

6. react-firebase-hooks

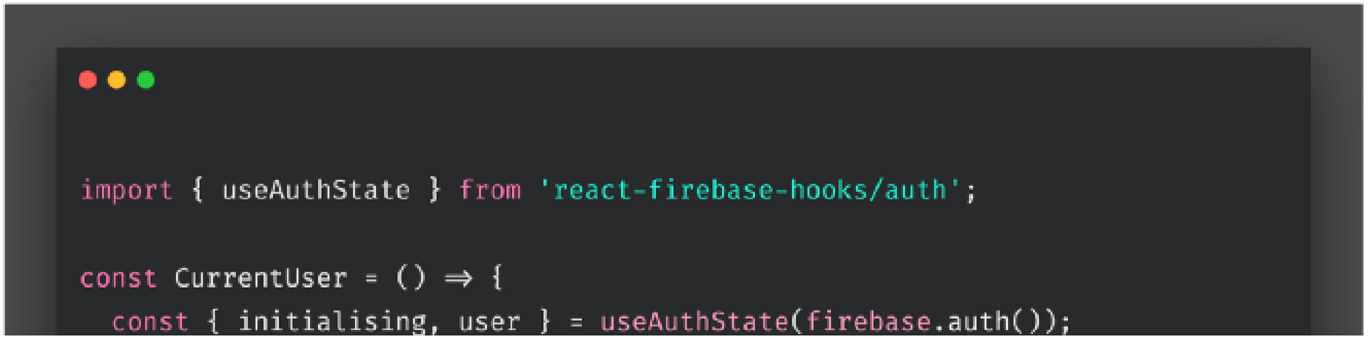
We all appreciate the greatness of firebase and use it a lot in our projects, whether its for authentication or storage. And this hook is the one we really need.

Installation:

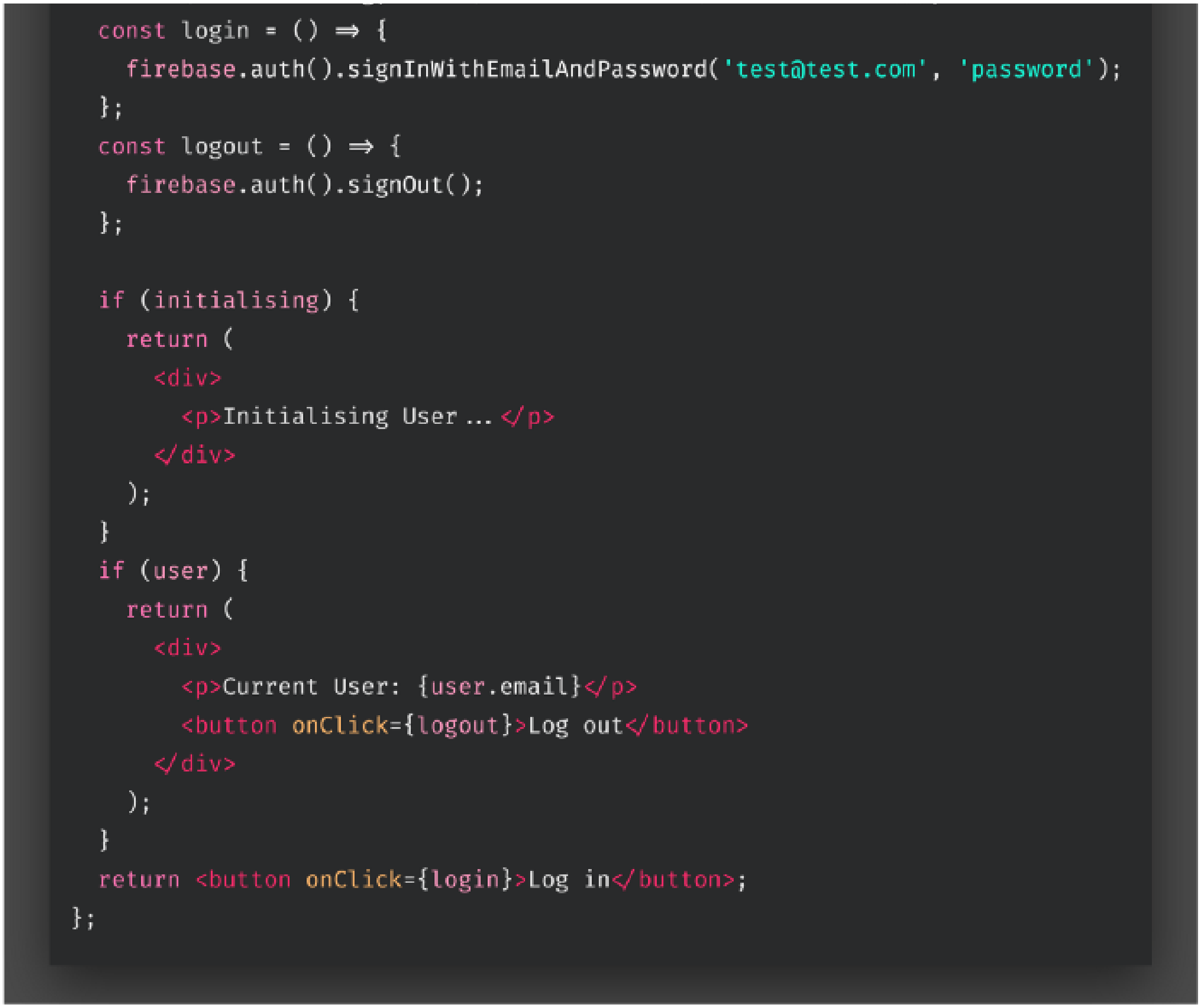
```
npm i react-firebase-hooks
```

Usage:

Below is the `useAuthState` hook, for authentication.



```
import { useAuthState } from 'react-firebase-hooks/auth';

const CurrentUser = () => {
  const { initialising, user } = useAuthState(firebase.auth());
```



```
  const login = () => {
    firebase.auth().signInWithEmailAndPassword('test@test.com', 'password');
  };
  const logout = () => {
    firebase.auth().signOut();
  };

  if (initialising) {
    return (
      <div>
        <p>Initialising User...</p>
      </div>
    );
  }
  if (user) {
    return (
      <div>
        <p>Current User: {user.email}</p>
        <button onClick={logout}>Log out</button>
      </div>
    );
  }
  return <button onClick={login}>Log in</button>;
};
```

useAuthState hook

The hook wraps around the `firebase.auth().onAuthStateChange()` method to ensure that it is always up to date.

Parameters: `auth` : `firebase.auth.Auth`

Returns: `AuthStateHook` containing:

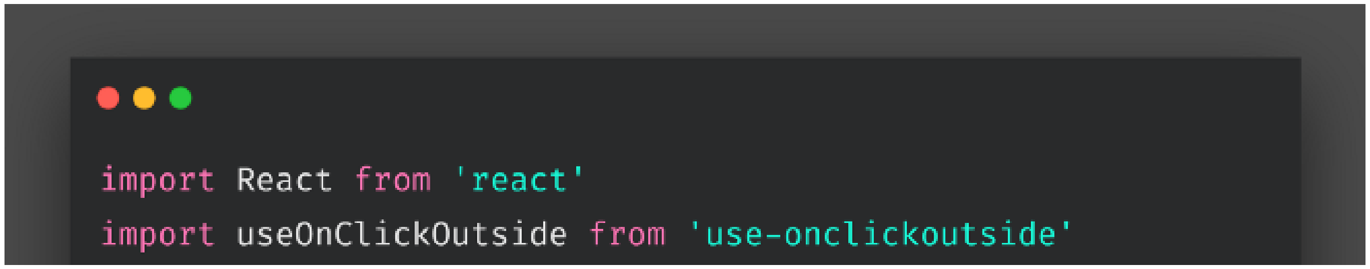
- `initialising` : If the listener is still waiting for the user to be loaded
- `user` : The `firebase.User` , or `null` , if no user is logged in

For a much detailed look, you can check out the useAuthState Bit component or the GitHub repo.

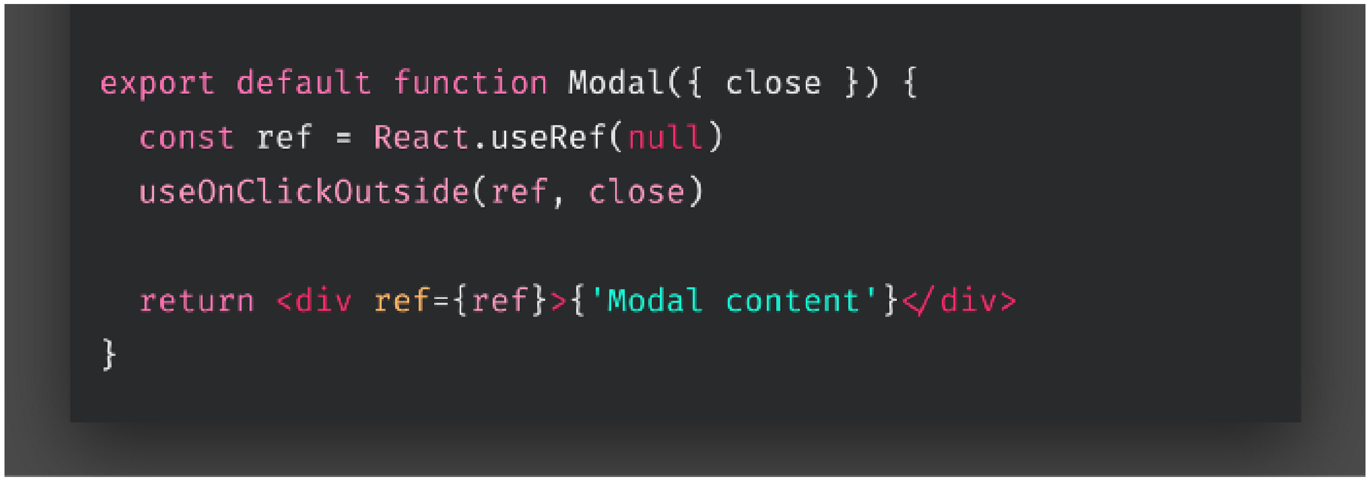
7. use-onClickOutside hook

An outside click is a way to know if the user clicks everything but a specific component. You may have seen this behavior when opening a dropdown menu or a modal or a dropdown list.

Usage:



```
import React from 'react'
import useOnClickOutside from 'use-onclickoutside'
```



```
export default function Modal({ close }) {
  const ref = React.useRef(null)
  useOnClickOutside(ref, close)

  return <div ref={ref}>{'Modal content'}</div>
}
```

useOnClickOutside hook

We provide the modal or component nodes to the ref and pass the ref inside our hook. If there is a click outside this modal then the close function runs.

For a much detailed look, you can check out the Bit component or the GitHub repo.

8. useIntersectionObserver hook

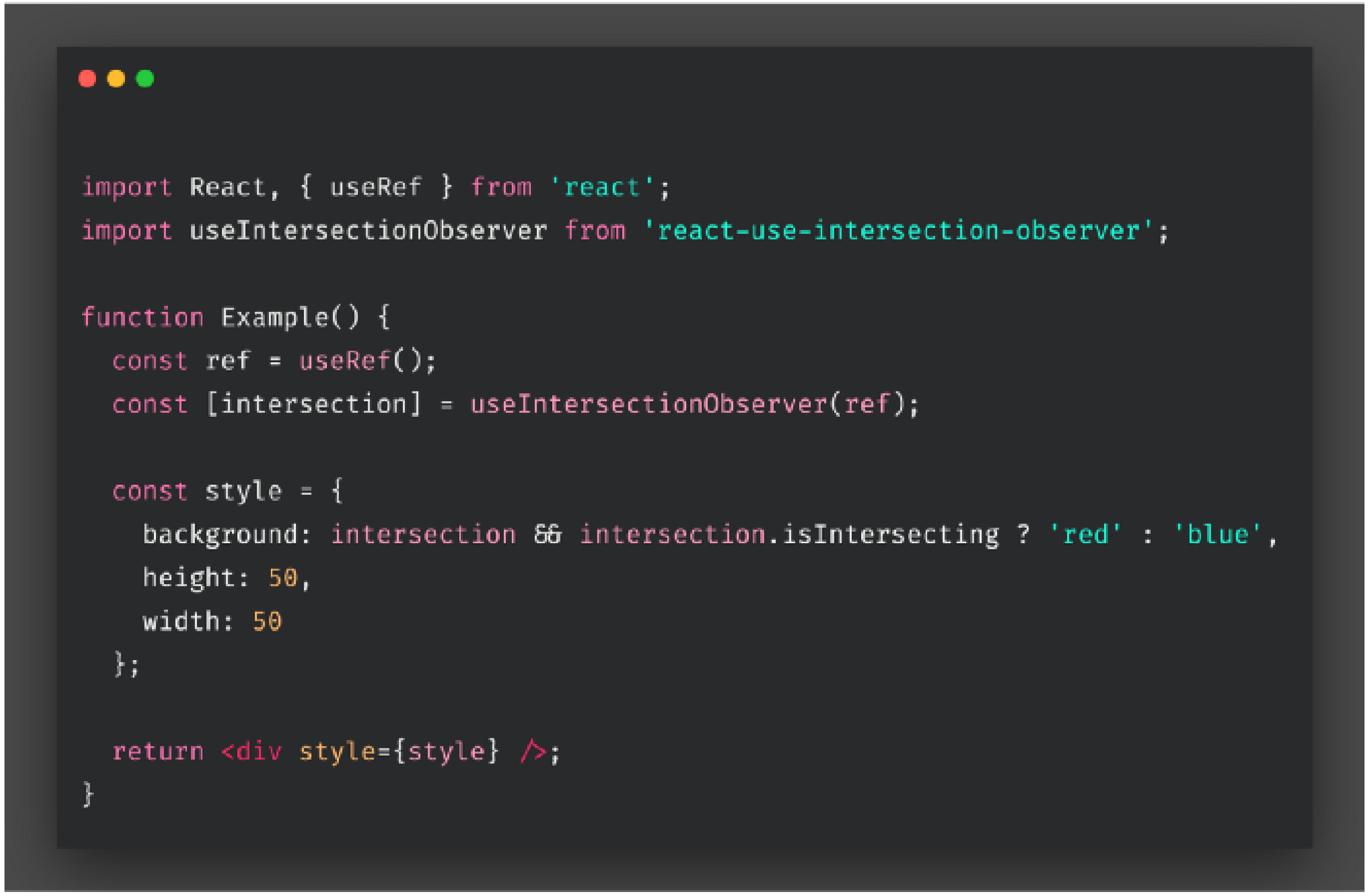
A React hook for using intersection observers.

The Intersection Observer API provides a way to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document's viewport.

Installation:

```
npm i react-use-intersection-observer
```

Usage:



```
import React, { useRef } from 'react';
import useIntersectionObserver from 'react-use-intersection-observer';

function Example() {
  const ref = useRef();
  const [intersection] = useIntersectionObserver(ref);

  const style = {
    background: intersection && intersection.isIntersecting ? 'red' : 'blue',
    height: 50,
    width: 50
  };

  return <div style={style} />;
}
```

useIntersectionObserver hook

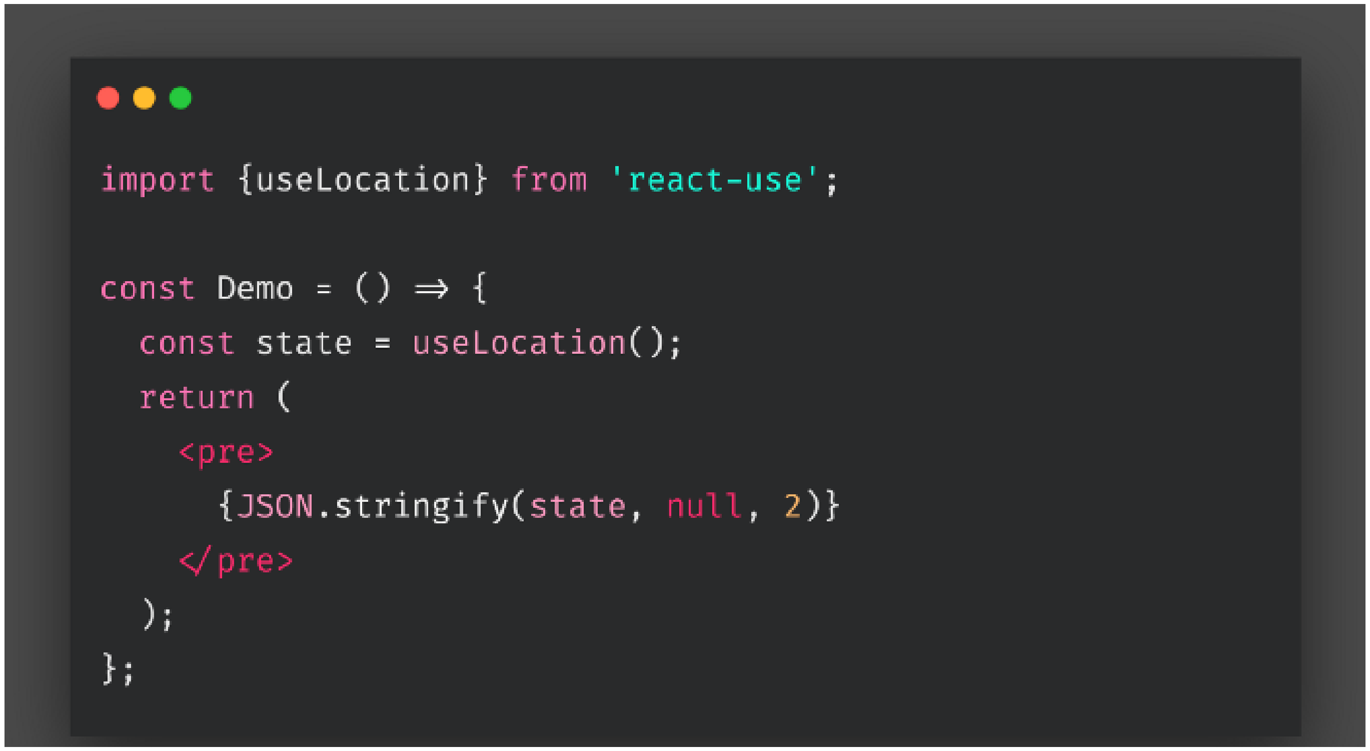
For a much detailed look, you can check out the Bit component or the GitHub repo.

9. use-location hook

The name says it all, this hook is used for getting the location of the browser.

Usage:

This hook is super useful and super easy to use:



```
import {useLocation} from 'react-use';

const Demo = () => {
  const state = useLocation();
  return (
    <pre>
      {JSON.stringify(state, null, 2)}
    </pre>
  );
};
```

useLocation

Check out the hook in much more detail with the Bit component, here.

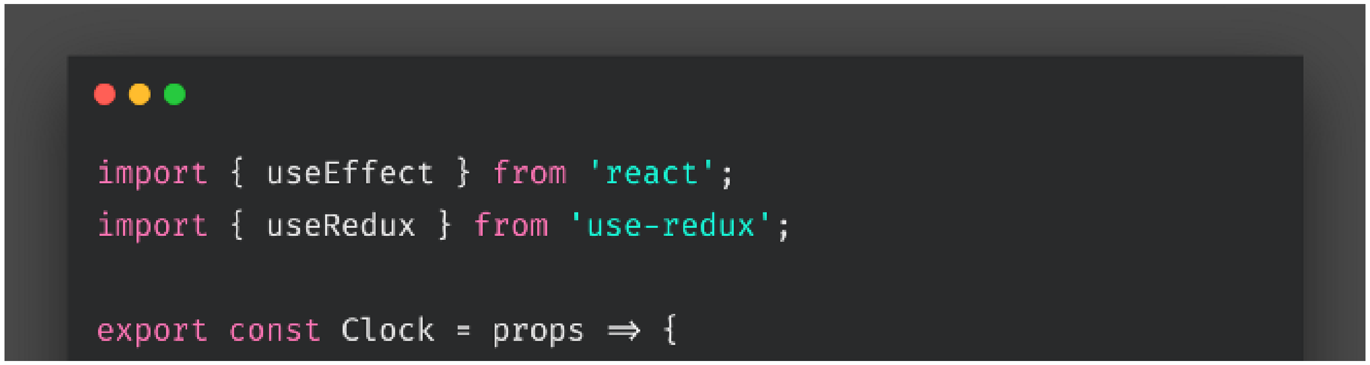
10. use-redux hook

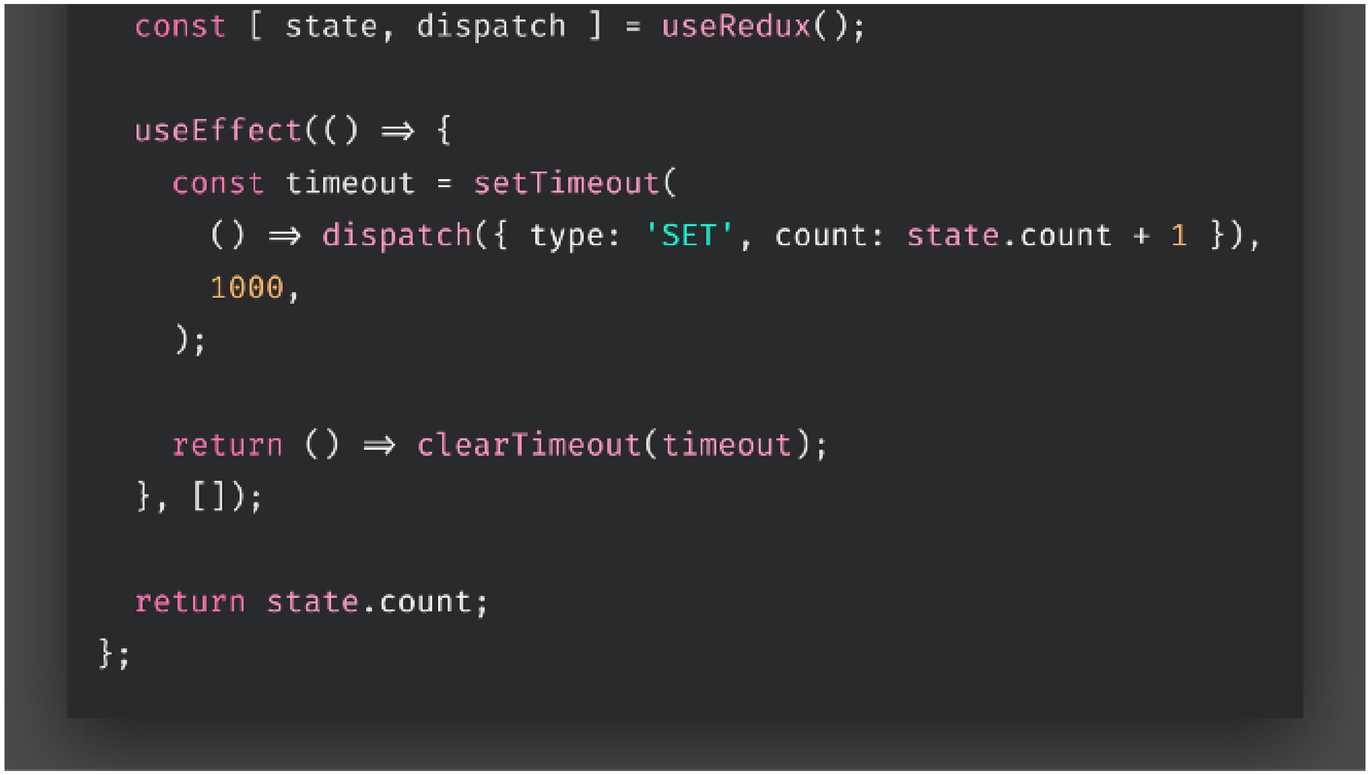
This one is for my redux readers. This hook returns the store and dispatch property.

Installation:

```
yarn add use-redux
```

Usage:


```
import { useEffect } from 'react';
import { useRedux } from 'use-redux';

export const Clock = props => {
```



```
  const [ state, dispatch ] = useRedux();

  useEffect(() ⇒ {
    const timeout = setTimeout(
      () ⇒ dispatch({ type: 'SET', count: state.count + 1 }),
      1000,
    );

    return () ⇒ clearTimeout(timeout);
  }, []);

  return state.count;
};
```

useRedux hook

The `dispatch` method is responsible for firing actions that cause changes in the store. There is a lot more to this so do check out the docs.

For a much detailed look, you can check out the Bit component or the GitHub repo.

Conclusion

I have created a Bit collection for these hooks, so feel free to check it out.

must-have-hooks-in-your-toolbox by aayush1408 · Bit

25 Javascript components. Examples: is-browsermacro , use-form-state , is-browser , constants , indexd , utils , index…

bit.dev

In this article, we discussed some cool stuff related to Custom Hooks. Hope you liked this article and learned something new and if you did clap your heart out and follow me for more content on Medium and as well as on Twitter. Please feel free to comment and ask anything.

I am pretty sure I missed out a lot of great custom hooks that are dwelling out there, so do tell me in the comment section what I missed :)

Thanks for reading 🙏 💖.

Learn more

5 Tools for Faster Development in React

5 tools to speed the development of your React application, focusing on components.

blog.bitsrc.io

11 React UI Component Libraries you Should Know in 2019

11 React component libraries with great components for building your next app's UI interface in 2019.

blog.bitsrc.io

Announcing Bit's Component Discovery Experience 2.0 ✨

We are excited to introduce a new experience for discovering and sharing your components. Take a look…

blog.bitsrc.io

React Hooks Frontend Development Web Development JavaScript