Using the Fullscreen API with React

Written by Seth Corker on Benevolent Bytes

The web is a powerful platform that has more to offer than you might expect. There are many APIs that enrich people's experience of the web allow developers to make websites that react in fun and interesting ways.

Using a browser in fullscreen isn't new but the web apps don't often tailor experiences for fullscreen. The Fullscreen API gives you the ability to adapt your web app based on whether the user is in fullscreen or windowed. Take advantage of the API in interesting ways to tailor the experience for users and take full advantage of the what your web browser has to offer.

What is it?

The Fullscreen API detects whether the browser is fullscreen or not and which element is in fullscreen. It also provides the ability to request fullscreen for a particular element and exit fullscreen.

Why is it useful?

You think of fullscreen mode as a browser specific feature, completely separate from the content the user interacts with. Fullscreen is accessible from the browser menu and a keyboard shortcut but it's also used in a few common places. There is often a fullscreen button on video elements but what can you do with it?

Presentation

If your web app can be used in a presentation context, it might be a good idea to make it easier for users to make their presentation fullscreen. The API is especially useful because you don't necessarily want to make the all of the UI visible in fullscreen. With the API, you could hide the editing elements and just show the presentation content.

Game

A web game is another good case of where the Fullscreen API could be useful. You could make the game area fullscreen instead of the whole web page or you may even want to adjust the UI to take advantage of the larger screen. If your app relies on common browser navigation then you could make your own when in fullscreen mode to ensure your app works well in all contexts.

How do I use it?

Creating a custom hook makes it easier to consume in a react app.

Helper method

Support across modern browsers is okay however we can increase support by using vendor prefixed properties for `fullscreenElement`.

```
function getBrowserFullscreenElementProp() {
  if (typeof document.fullscreenElement !== "undefined") {
    return "fullscreenElement"
  } else if (typeof document.mozFullScreenElement !== "undefined") {
    return "mozFullScreenElement"
  } else if (typeof document.msFullscreenElement !== "undefined") {
    return "msFullscreenElement"
  } else if (typeof document.webkitFullscreenElement !== "undefined") {
    return "webkitFullscreenElement"
  } else {
    throw new Error("fullscreenElement is not supported by this browser")
  }
}
```

The hook

I created a custom `useFullscreenStatus` hook which accepts the ref of the element we want to make fullscreen. It returns an array containing a boolean representing whether the element is fullscreen and a function to set

the element to fullscreen. There are a couple of things to note. `requestFullscreen()` can be called on html elements and returns a promise. The hook also sets `document.onfullscreenchange` to detect if we enter fullscreen on any element, `useLayoutEffect` is used instead of the usual `useEffect` because when the we enter fullscreen, it's not instant and `useEffect` failed to detect the changes - possibly because the DOM needed to be updated first.

Using the hook

The hook can be consumed in a similar way to the `setState` hook.

```
const [isFullscreen, setIsFullscreen] = useFullscreenStatus(maximizableElement)
```

If the Fullscreen API isn't supported by the browser then the helper will throw an error. There are a couple of ways to deal with this, in the example I used the following:

```
try {
  ;[isFullscreen, setIsFullscreen] = useFullscreenStatus(maximizableElement)
} catch (e) {
  errorMessage = "Fullscreen not supported"
  isFullscreen = false
  setIsFullscreen = undefined
}
```

If your component needs the Fullscreen API, it may be better to detect support and conditionally render the component.

The Fullscreen API is useful for specific use cases, you might not always need it but it could be useful for delivering more engaging experiences in your web apps.

Resources

- Fullscreen API on MDN
- Want the source for the demo? Take a look at the repository on GitHub.