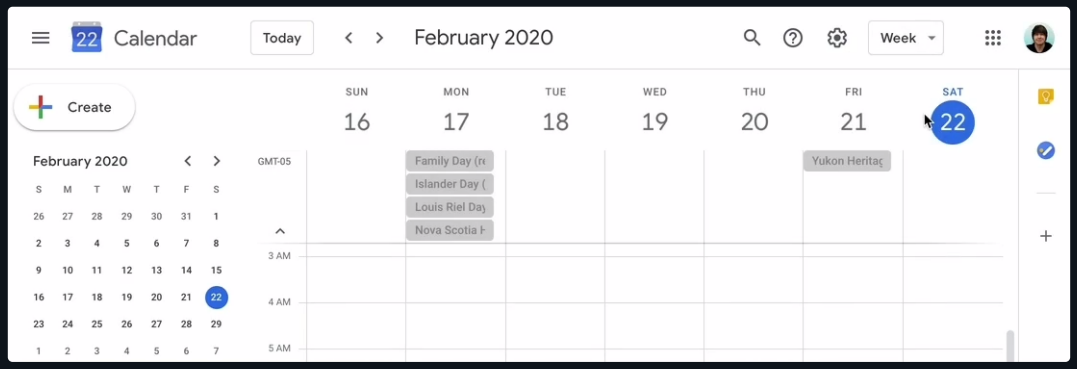
Persisting React State in localStorage

Introducing the “useStickyState” hook

Let's say we're building a calendar app, like Google Calendar. The app lets you toggle between three different displays: month, week, and day.



Toggling between views in a typical calendar application

Personally, I always want to see the "Week" view. It gives me everything I need to know about the current day, while also giving me a peek at what's coming

to know about the current day, while also giving me a peek at what's coming up in the next couple of days.*
Thankfully, calendar apps know that users have strong preferences around this kind of thing, and the toggle is *"sticky"*. If I switch from "week" to "month" and refresh the page, the "month" view is the new default; it sticks.

Conversely, it's **super annoying** when form controls aren't sticky. For example: every month, I create 4-5 expenses through Expensify. Every single time, I have to swap the default currency from USD to CAD. Why can't it remember that I'm Canadian??

In this tutorial we'll see how we can create a *custom React hook* to abstract away the "stickiness", so we get it for free whenever we need it.

Show me the code

Here's what our custom hook looks like:

JS

```
function useStickyState(defaultValue, key) {
  const [value, setValue] = React.useState(() => {
    const stickyValue = window.localStorage.getItem(key);

    return stickyValue !== null
      ? JSON.parse(stickyValue)
      : defaultValue;
  });

  React.useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);
```

```
  return [value, setValue];
}
```

What about SSR?

If your app is server-rendered (with a framework like Next.js or Gatsby), you'll get an error if you try using this hook as-is.

This is actually a pretty tricky problem, because that first render on the server doesn't have access to your computer's localStorage; it can't possibly know what the initial value should be!

Dynamic content in a server-rendered app is a complex subject, but fortunately, I've written a blog post on exactly this topic! Read **The Perils of Rehydration** to learn more.

To show how it works, here's a quick counter demo with a sticky count. Try clicking it a few times, and then refresh this page:

JSX

```
function App() {
  const [
    count,
    setCount
  ] = useStickyState(0, "count");

  return (
    <div className="App">
      <h1>Counter</h1>
      <p>Current count: {count}</p>
```

Counter

```
      <button
        onClick={() => setCount(count + 1)}
      >
        Increment
      </button>
    </div>
  );
}

render(<App />);
```

Current count: 0

Increment

Enable 'tab' key

If this code isn't clear to you, fear not! The rest of this tutorial explains it in greater detail 💫

In practice

This hook makes a single assumption, which is reasonably safe in React apps: the value powering a form input is held in React state.

Here's a non-sticky implementation of a form control to switch between values:

JSX

```
const CalendarView = () => {
  const [mode, setMode] = React.useState('day');

  return (
```

```
    <>
      <select onChange={ev => setMode(ev.target.value)}>
        <option value="day">Day</option>
        <option value="week">Week</option>
        <option value="month">Month</option>
      </select>

      {/* Calendar stuff here */}
    </>
  )
}
```

We can use our new "sticky" variant by swapping out the hook:

JS

```
const CalendarView = () => {
  const [mode, setMode] = useStickyState('day', 'calendar-view');

  // Everything else unchanged
}
```

While the `useState` hook only takes 1 argument—the initial value—our `useStickyState` hook takes two arguments. The second argument is the key that will be used to get and set the value persisted in localStorage. The label you give it has to be unique, but it otherwise doesn't matter what it is.

How it works

How it works

Fundamentally, this hook is a wrapper around `useState`. It just does some other stuff too.

Lazy initialization

First, it takes advantage of lazy initialization. This lets us pass a function to `useState` instead of a value, and that function will only be executed the first time the component renders, when the state is created.

JS

```
const [value, setValue] = React.useState(() => {
  const stickyValue =
    window.localStorage.getItem(key);

  return stickyValue !== null
    ? JSON.parse(stickyValue)
    : defaultValue;
});
```

In our case, we're using it to check for the value in localStorage. If the value exists, we'll use that as our initial value. Otherwise, we'll use the default value passed to the hook ("day", in our earlier example).

Keeping localStorage in sync

The final step to this is to make sure that we update localStorage whenever the state value changes. For that, our trusty friend `useEffect` comes in

handy:

```js
React.useEffect(() => {
  window.localStorage.setItem(name, JSON.stringify(value));
}, [name, value]);
```

Rapid-fire updates?

If the state value changes rapidly (like, many times a second), you may wish to throttle or debounce the updates to localStorage. Because localStorage is a synchronous API, it can cause performance problems if it's done too rapidly.

Don't take this an excuse to prematurely optimize, though! The profiler will show you whether or not your updates need to be throttled.

Wrapping up

This hook is a small but powerful example of how custom hooks let us invent our own APIs for things. While packages exist that solve this problem for us, I think there's a lot of value in seeing how to solve these problems ourselves 🧙‍♂️

Special thanks to Satyajit Sahoo for a couple refactor suggestions 🌟

LAST UPDATED

February 24th, 2020

HITS

A front-end web development newsletter that sparks joy

My goal with this blog is to create helpful content for front-end web devs, and my newsletter is no different! It includes **early previews** to upcoming posts and access to special bonus goodies. No spam, unsubscribe at any time.

First Name

Email

Subscribe

Josh Comeau

Thanks for reading!

/