React Animation - Tap to Expand

Written by Seth Corker on Benevolent Bytes

What we're making

We'll be making a card that expands when you tap on it. The example is a horizontal list of cards you can scroll through (achieved with CSS `scroll-snap-type`). When you tap or click on the card it expands to take up the whole screen and prevents scrolling.

A demonstration video of the tap to expand animation You can play with the demo directly or take a look at the repo.

If you'd like to see a video tutorial, I go through the whole process of adding the animations and making adjustments to get the desired effect.

A walkthrough of the complete process of adding the tap to expand animation states

Tips for web animation

When adding animation with Framer Motion, other animation libraries in React and JS in general, you may need to change the structure of the HTML. Sometimes, it might be easier to add wrapping divs and animate those rather than the existing elements. If you find yourself fighting to try and get something animating, take a step back and see if you can make a trade-off. Adding a wrapper dilutes the structure of your HTML a bit but might make your animation code much cleaner. I prefer to use `div` elements because you're adding it just to manipulate in code and it doesn't need a semantic meaning.

How to animate with the useCycle hook

Framer Motion comes with a few useful React hooks, for our tap to expand animation, we can achieve the effect with the `useCycle` hook. It provides us with the current state and a function to call when we want to progress the state. To build the hook, supply a list of arguments. In our case, we can use two objects representing the two different states we need to animate between.

```
const [animate, toggleFocus] = useCycle(
  { height: "25rem", top: "0rem", overflowX: "auto" },
  { height: "100%", top: "-4.4rem", overflowX: "hidden" }
)
```

The useCycle definition The state will start with `height: "25rem"`, etc. When we call `toggleFocus` the first time, the state will change to `height: "100%"`, etc. If we continue to call the `toggleFocus` function, `useCycle` will infinitely cycle through these states.

To animate, we use the `motion` component and pass in our `animate` state. Motion will handle this directly and animate between the props which change. To trigger the animation, we'll use the `onTap` prop (also provided by the motion `component`) and pass it the function from `useCycle`.

```
<motion.div onTap={() => toggleFocus()} animate={animate} />
```

Consuming the data from our useCycle hook What makes this hook very versatile is that we can supply any number of arguments and they can be of any type. If there are multiple states we need to move between linearly, this hook works well. The cycle function also accepts an index if so we can navigate between the different UI states in a non-linear manner. For our animation, we just need two states but if we wanted to control more than one, it's possible to supply more complex objects.

This is a snippet of one of the `useCycle` hooks used in the demo. Using complex objects allows for the states and animations of multiple elements on the screen to animate in sync when `cycleCard` is called.

```
const [animate, cycleCard] = useCycle(
  {
    card: { padding: "1rem" },
    image: {
      width: "100%",
      marginLeft: "0rem",
      marginRight: "0rem",
      marginTop: "0rem",
```

```
    },
  },
  {
    card: { padding: "0rem" },
    image: {
      width: "125%",
      marginLeft: "-3rem",
      marginRight: "-3rem",
      marginTop: "-1rem",
    },
  }
)
```

Where to go from here?

In the demo, we transition between two different states, expanded and collapsed. To simplify the code, the states could be moved into variants (as described in *"A simple loading animation with Framer Motion"*). I recommend experimenting with the `useCycle` hook where you can clearly define the different states the UI needs to be in. It's easy to set up and understand while being a very powerful system for controlling animation.

Resources

- To see the full source code, check out the repo on GitHub
- Check out my playlist of video tutorials covering animation in Framer Motion
- Take a look at the official Framer Motion documentation