



To make Medium work, we log user data. × [Learn more](#)
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.

Open in app

Get started



Joe Davis

Follow

Jul 1, 2016 · 8 min read · Listen

UI Animations with React — The Right Way

When building web application UIs, I've always been a big fan of transitions. Instead of just popping things in and out, everything that moves needs to fade or slide in and out of existence, bringing the super polished and professional look. [jQuery](#) made this extremely easy, and for a long time, it was the standard for bringing that web 2.0 look to your pages.

Then came the front-end frameworks like [Knockout](#), [Backbone](#), [Angular](#) and [React](#), and suddenly it wasn't cool to use jQuery to manipulate the DOM anymore because that was the job of your shiny new framework. If any changes get made to the UI, they should be the result of the underlying data model changing and the framework altering the UI to reflect those changes. So jQuery began to take a back seat as it violated this principle.

If you're like me, when you first started using a front-end framework, you probably came up with some pretty creative ways to implement your precious animations. When I first started using React, I was immediately sold on how easy it was to build very complicated UIs in a simple and straightforward way. It was powerful, scalable, fast and fun. However, I noticed that it didn't seem to lend itself well, at least at that time (2015), to UI transitions.

Here is an example of my initial strategy for implementing a fade with React.

```
1  import React from 'react';
2  import {findDOMNode} from 'React-dom';
3
4  class ExampleComponent extends React.Component{
5      constructor (props) {
6          super(props);
7      }
8  }
```





To make Medium work, we log user data.
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.



Open in app

Get started

```
14      $(findDOMNode(this)).hide().fadeIn('slow');
15    }
16  }
17 }
18
19 render() {
20   return (
21     <div className="example-component"></div>
22   );
23 }
24 }
25
26 export default ExampleComponent;
```

view raw

In the `componentWillReceiveProps` life-cycle method I'm checking the `visibility` prop and then using jQuery to fade the component in or out based on the value of the prop. And this works, but... yuck!

I rationalized that this was a good solution because, even though I am using jQuery to manipulate the DOM (big no no), I was doing it in response to a data change. So I'm following the rules even though I'm breaking the rules. This is not an ideal solution in at least 3 other ways.

1. Although this works in this context, you often will run into problems in more complicated scenarios. For example: if you want to fade something out before removing it completely from the DOM, you would have to wait for the completion of the fade before you update the data which would in turn unmount the component. This is manageable, but it generally introduces more complication than you would want just to add some UI sugar.
2. jQuery isn't exactly a light library. It clocks in at around 85k minified. Tossing that extra weight into your client download is costly, especially when there are better options available. (read on)
3. With the advent of stateless functional components, the component life-cycle methods including `componentWillReceiveProps` that is used in this example, are no longer available. Stateless functional components encourage good coding practices,





To make Medium work, we log user data.
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.



Open in app

Get started

We can achieve the same [transitions](#).
ent using [CSS](#)





To make Medium work, we log user data.
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.



Open in app

Get started

In this example component, a single boolean value is being passed in as a prop to determine if the 'show' CSS class should be added to the component. As the value is flipped between true and false, the class will be added and removed from the component root element. Then we simply need to rely on the power of CSS transitions. The example-component class begins invisible, but when we add the additional class of show, it will animate a fade-in.

Note that the transition is taking place over the 'opacity' and 'visibility' CSS rules as 'display' does not animate and will simply snap the element in and out. Opacity is responsible for the fade and visibility is responsible for removing click and hover events.

Problems with CSS Transitions

This method isn't without its own nuances as well. First and possibly the most obvious is that CSS transitions are a relatively new feature and are not supported in older browsers. This however could be considered a non-issue as non-supporting browsers will fail gracefully. A browser that does not support the transition style rules will simply





To make Medium work, we log user data.



By using Medium, you agree to our

[Privacy Policy](#), including cookie policy.

Open in app

Get started

Another problem you may encounter is the content being behind where the content had been prior to the fade animation as the 'visibility:hidden' style rule does not actually collapse the content. If this ends up being a problem, you can reduce the height or width of the element to zero as part of the transition. As doing so may alter the arrangement of the content during the animation, you may want the fade out animation to complete before size reduction animation begins. This was easily accomplished with jQuery using callbacks. But callbacks don't exist in CSS so an alternate means has been provided: transition-delay.

Using transition delays to order the animations one way in and the other way out can be a little tricky.





To make Medium work, we log user data.



By using Medium, you agree to our

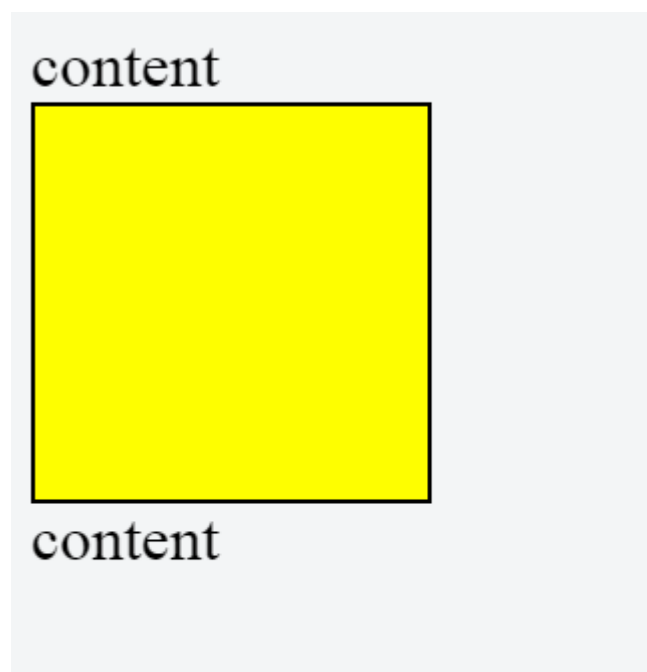
[Privacy Policy](#), including cookie policy.

Open in app

Get started

In this example we have a box with a transition rule. When you want the content to fade out you add the hide class. As soon as the hide class is applied, the transition rule from that class overwrites the original one in the box class and it will be the transition rule that is followed for the altered properties of the hide class.

The 'transition' rule will take a comma separated list of CSS properties you wish to animate. For each property you wish to animate you put the property name, the transition-duration, and then the transition-delay. Note that in the transition rule in the hide class, both height and opacity have a 500ms duration, but height has a 500 ms delay where opacity has no delay. This will cause the height animation to wait until after the opacity animation is complete. When we remove the hide class the transition rule immediately changes to the one listed originally in the box class, wherein the opacity has the 500ms delay and height has no delay. Thus the order in which the animations execute is reversed. Like I said, this can be tricky, especially if you try to get overly complex with the number of things you are trying to animate at different times. You can see a short clip of this working below.




OK, Awesome! But there is still one more problem we need to cover.

When working with React, there are times when you want to animate a component





To make Medium work, we log user data.  [Learn more](#)
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.

Open in app

Get started

have been added to the array. When an item is removed from the array. This can be problematic because as soon as you alter the underlying data in the array by removing an item, React will unmount the associated component immediately before you have a chance to fade it out. Likewise, if you are using stateless functional components (which we've already established is a good idea whenever possible) you have no life cycle methods to trigger the fade-in animation when a component is first mounted as new objects are added to your array.

Luckily, the folks over at facebook have realized this and have blessed us with a new add-on, [ReactCSSTransitionGroup](#), to solve the problem. [ReactCSSTransitionGroup](#) is itself a component in which you can nest child components. Any child component that mounts will undergo the 'enter' transition right after it mounts, and any child component that unmounts will undergo a leave transition before it unmounts. The [ReactCSSTransitionGroup](#) component takes in `transitionEnterTimeout` and `transitionLeaveTimeout` as props. What these values represent are the duration in milliseconds of your enter and leave transitions. Essentially these props help [ReactCSSTransitionGroup](#) know when to add and remove the various CSS classes involved in the transitions and when it can unmount child components.





To make Medium work, we log user data.
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.



Open in app

Get started

The CSS classes that ultimately cause the animations need to be defined by hand. The class names that will be applied by `ReactCSSTransitionGroup` are defined by convention based on the value passed in to the `transitionName` prop. The values for the two timeout props should correspond to the transition durations in the CSS classes that you define.





To make Medium work, we log user data. ×
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.

Open in app

Get started

By convention, if the value of the `transitionName` prop of `ReactCSSTransitionGroup`, it will look for these CSS classes:

- `example-enter` : defines the beginning state of the enter transition
- `example-enter.example-enter-active` : defines the actual enter transition
- `example-leave` : defines the beginning state of the leave transition
- `example-leave.example-leave-active` : defines the actual leave transition

Once you get it wired up, the result is really smooth and satisfying.

Add Item



That about Sums it up!

With this set of tools, you should be able to create just about any transition you can think of in a React application, and you'll be doing it without breaking the rules of modifying the DOM manually.

Addendum

Since the time I wrote this post, `ReactCSSTransitionGroup` has been deprecated and is no longer supported. It's continued use is not recommended. However, the component has been moved to the third party package, [react-transition-group](#), which is now supported by the community and still functions exactly the same as described above in this post.

```
// So instead of this
import ReactCSSTransitionGroup from 'react-addons-css-transition-group';
```





To make Medium work, we log user data. ×
By using Medium, you agree to our
[Privacy Policy](#), including cookie policy.



Open in app

Get started

I've also been asked if the [react-motion](#) library is still relevant today as this post was written almost two years ago. There are a number of very cool libraries available now that allow you to do some pretty fancy stuff. But the vast majority of anything you would need to do in a standard web application can be achieved with the methods I've outlined above and without the overhead of including large libraries. They are none the less worth a look. Two of the most relevant ones are [react-motion](#) and [react-spring](#).

I'd be happy to answer any questions. Hit me up in the comments below or find me on Twitter [@JoeTheDave](#).

. . .

Need a little extra help getting that React project moving? I've been developing in React for years. I can help transition a team to React or just jump in as another senior level dev to help drive the project forward. I'd love to help. Let's talk. Message me at joe@sage-development.net.

