



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



Published in React Native Training

You have **2** free member-only stories left this month.

[Sign up for Medium and get an extra one](#)



Nader Dabit

Follow

Aug 6, 2017 · 11 min read ★ · Listen

React Animations in Depth

5 Options for React Web Animations, and which of these are cross-platform.

In this post, we will look at the following ways to create React animations:

1. CSS Animations based on React component state
2. JS Style animations based on React component state
3. React Motion by **Cheng Lou**
4. Animated
5. Velocity-React

To view the repo for this project, click [here](#).

Bonus: At the end, I will also show how to implement animations using [GreenSock](#) as it is something I was familiar with, but I'm not sure I would recommend it yet as I have yet to test performance with React but it seems to work great!!

Edit: Concerning GreenSock, they tweeted out a link to check out concerning perf:

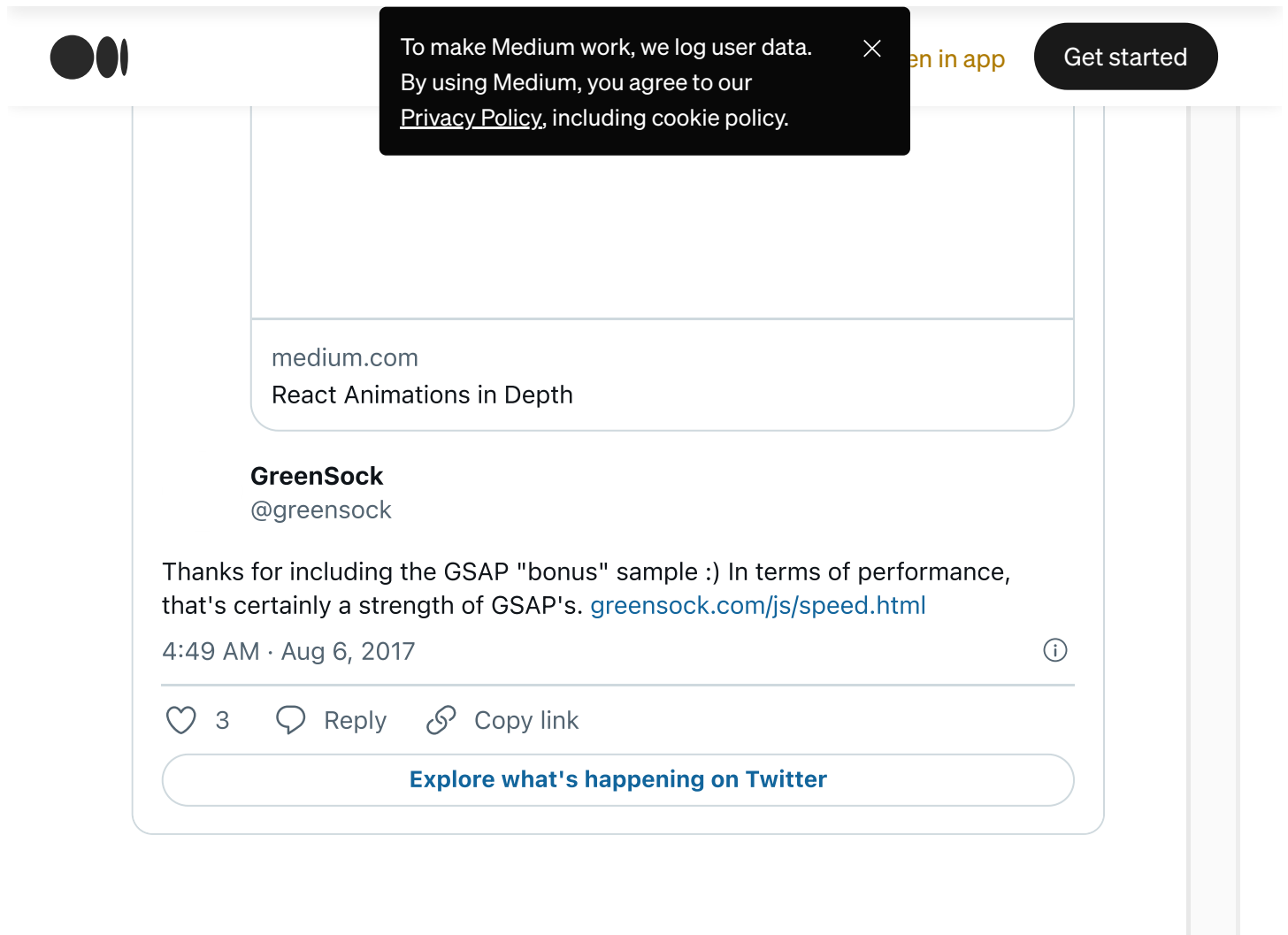
nader dabit (🧱, 🚀) | sha.eth | nader.sol · Aug 6, 2017

@dabit3

I just posted React Animations in Depth on a Saturday night 🤔

[medium.com/react-native-t...](#) #react #reactnative





A few weeks ago I was working on a project and I wanted to know the current best practices for animations using React web, so I sent out a tweet and got some good responses.

Because I have been head down into React Native for so long, I was somewhat out of touch with the current best practices for some things web related





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

got into React programming

What I realized is there are quite a few ways to do animations in React these days, so I explored many of the options and decided that documenting what I learned here would be great for anyone looking to view their options when it comes to this subject.

Let's get started!

. . .

1. CSS Animations based on React component state

The most basic and obvious way for me to animate was using CSS class properties and adding / removing them to perform an animation. This is a good way to do basic animations if you are already using CSS in your app.

The benefits of doing your animations this way are definitely performance.

Cons: The drawbacks of this approach is that it is not cross-platform (won't work with React Native) , it relies on CSS and the DOM, and if you needed to create something complex, it would be hard to control using this method.

Pros: Regarding CSS animations, there is somewhat of a known rule that if you don't change any properties besides opacity or transform, you will usually have great performance. Updating these values based on state is very simple, and will give us smooth transitions with only a single rerendering of our component.

Let's take a look at an example of this. We will animate an **input** using CSS animations along with React component state:





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

First, let's go ahead and

...ing to our input.

We have a basic input with some styling, and we have set the transition to `width .35 linear`, giving the animation some properties for us to have in place.

We also have an input-focused class that will change the width from 150 pixels to 240 pixels.

Now, let's wire this up to work in our React app:





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

1. We create an initial state of `focused` and set it to false. We will toggle this to update our animated component.
2. In `componentDidMount`, we add two listeners, one for blur and one for focus. Both of these listeners will call the focus method we will highlight next. Notice we are referencing `this.input`, this is because we use the `ref` method on the input to create a reference to this input and set it as a class property. We do this in `componentDidMount` because we do not have access to the dom in `componentWillMount`.
3. The `focus` method will check for the previous value of `focused`, and toggle based on its value.





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

`input-focused` class.

have a usable `className`.

and call `.join(' ')` to

2. JS Style animations based on React component state

The way we will create animations using JS styles is very similar to the way we did this using CSS classes. The benefit here is that you get the same performance but you do not have to resort to CSS classes, you can have all of your logic in your JS file.

Pros: Like CSS animations, the benefits of doing your animations this way are definitely performance. It also is a good approach because you do not have to rely on any CSS.

Cons: Like CSS animations, the drawbacks of this approach is that it is not cross-platform (won't work with React Native), it relies on CSS and the DOM, and if you needed to create something complex, it would also be hard to control.

In this example, we will create an input that when typed into, will turn a button from disabled into enabled, and give user feedback as to the button now being enabled.





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

1. We create an initial state of disabled, set to true.
2. The `onChange` method will be attached to the input, and we will check to see how many characters have been typed. If there are 4 or more, we will set the state of disabled to false, and if not we will set it to true if it is already not set to true.
3. The style property of the button element will determine whether or not we add the animated class, `buttonEnabled`, to the element based on the value of `this.state.disabled`.
4. The style of button has a transition of `.25s all` because we want to animate both the `backgroundColor` as well as the `width` properties of the element.

3. React Motion

React Motion is a great library written by [Cheng Lou](#), who has worked on Animations for over two years, both for React Web and React Native. [Here](#) is a great talk where he discusses animations at React Europe in 2015.





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

may need to cancel or stop working with variable dimensions in your application.

The idea behind React Motion is that you set a style configuration on a React Motion component, and you receive a callback function with children that represent those style values. Here is a basic example of what this looks like:

```
<Motion style={{ x: spring(this.state.x) }}>
  {
    ({ x }) =>
      <div style={{ transform: `translateX(${x}px)` }} />
  }
</Motion>
```

Pros: React Motion can be used in React Native as well as React Web because it is cross-platform. The `spring` concept felt weird to me at first, but after actually using it is genius and handles everything really well. It has a very nice API as well!

Cons: I've noticed that it is not as performant as pure CSS / JS styling animations in some cases. The API is something new you will have to learn, although it is pretty easy to pick up!

To use this library, you will have to install it via npm or yarn:

```
yarn add react-motion
```

In this example, we will create a dropdown menu that will expand and contract on a button press.





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


Animate

Selection 1

React Motion





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

Get started

1. We import `Motion` and `spring` from `react-motion`.
2. We set an initial state of `height` set to 38. We will use this to animate the height of the menu.
3. The `animate` method will check to see the existing height value set on the state, if it is 38 it will change it to 250, and if not it will set it back to 38.
4. In Render, we use the Motion component to wrap a list of `p` tags. We set the style property of Motion, passing the current value of `this.state.height` as the `height` value. Now, height is available as a child in the callback within the Motion component. We use the value in the callback to declare the height of the parent `div` wrapping the list of styles.
5. When the button is clicked, we toggle the animated height property by calling `this.animate`.

4. Animated

The Animated library is based off of the same Animated library that is being used in React Native.

The basic idea behind Animated is you can create declarative animations, and pass in configuration object that control what is going on within your animation.

Pros: Cross Platform. It is already very stable in React Native, so if you learn it there you do not have to learn it again. Animated allows us to interpolate a single value into multiple styles using the `interpolate` method (implemented below). We can also take





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

It is a fun to use API once you have everything set up. You can do many things and have a lot of control and flexibility over your animations. There is also much documentation around it, including many [blog posts](#) and [videos](#) about it by [Jason Brown](#).

For a detailed look into using Animated specifically with React Native, check out [this](#) blog post.

Cons: Based on the exchange I had via Twitter, it seems like it is not 100% stable on the web yet, with issues like auto-prefixing for older browsers as well as some performance issues. It is also a new API that you will have to learn if you do not already know it from React Native.

To use this library, you will have to install it via npm or yarn:

```
yarn add animated
```

In this example, we will simulate an animated message that will show up after a form submission is complete.




Animate

Animated confirmation using the Animated library.





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

1. Import both `Animated` and `Easing` from `animated`. Notice that we do not import the entire library directly, but we actually reference the `react-dom` and `Easing` APIs directly.
2. Create a class property of `animatedValue`, and set it to zero. We do this by calling `new Animated.Value(0)`.
3. Create an `animate` method. *This method will handle all of the animation that will be taking place, though we will use this animated value later and create other animated values off of it using the `interpolate` method.* In this method, we set the animated value to zero by calling `this.animatedValue.setValue(0)` so this animation will be triggered every time this function is called. We then call `Animated.timing`, passing





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

time of the animation we would like (we chose `Elastic`).

4. In our render method, we first create a new value called `marginLeft` by using the `interpolate` method available on any animatable value. Interpolate takes a configuration object with an `inputRange` array, and an `outputRange` array, and will create a new value based on the input and output. We will use this value to set the `marginLeft` property of the animated message in our UI.
5. Instead of using a regular `div` , we use an `Animated.div` .
6. We style the `Animated.div` using both our `animatedValue` as well as our newly created `marginLeft` property, setting the `opacity` directly to the `animatedValue` , and `marginLeft` to `marginLeft` .

5. Velocity React

Velocity React is based off of the existing Velocity DOM library.

After using it, my perception is that it has an API that can be described as a combination of Animated and React Motion. Overall, it seems like an interesting library and I will keep it in mind when doing animations on the web, but overall I think I like React Motion and Animated more.

Pros: Very easy to get up and running with. The API is pretty simple and straightforward, and was easier for me to grasp at first than React Motion.

Cons: There were a few quirks that I had to get over when learning it, including the fact that the animation does not run on `componentDidMount` , and to do so you have to declare a `runOnMount` property on the animation. Also, this library is not cross-platform.

The basic API looks something like this:

```
<VelocityComponent  
  animation={{ opacity: this.state.showSubComponent ? 1 : 0 }}  
  duration={500}  
>
```





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

To use Velocity React, we

```
yarn add velocity-react
```

In this example, we will create a cool animated input that will animate the letters as we type them!



Velocity React





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

1. Import `VelocityComponent` from `velocity-react`.
2. We create a reusable component to hold each individual letter we will be animating, since we will be animating these one at a time.
3. In this component, we set the animation to an `opacity` of `1`, and a `marginTop` of `0`. These values represent the overriding values of the child component that we will be passing in, in this case a `<p>` with a beginning `opacity` of `0`, and `marginTop` of `100`. What will happen is that when the components are created, we animate the `opacity` from `0` to `1`, and the `marginTop` from `100` to `0`. We also set a duration of 500 milliseconds, and property called `runOnMount`, declaring that we want the animation to run when the component is mounted or created.
4. We have an `onChange` method hooked up to the input element in the render method. `onChange` will take each individual letter in the `input`, and create a new array using the `VelocityLetter` component we created at the top!
5. In render, we use this array to render the letters to our UI.

Conclusion

Overall, I think I will be using JS style animations for basic animations, and React Motion for anything crazy on the web. And for React Native, I will be sticking to





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

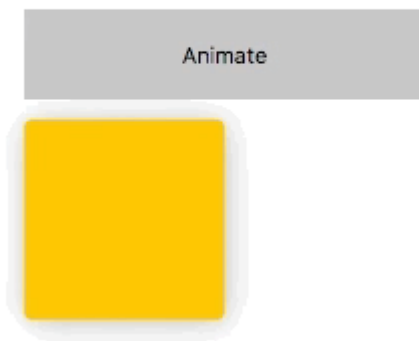
I mentioned at first that [this](#) also have a demo of how you may implement GreenSock animations in React (thanks in part to [this](#) discussion).

The great thing about GreenSock is it gives us the ability to easily chain animations, creating a complex sequence fairly easily!

To use GreenSock, we must first install it using yarn or npm:

```
yarn add gsap
```


Here, we will just create a box that animates in a square.



GreenSock Animation





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

Get started

The API for GreenSock is great, and is located [here](#).

My Name is [Nader Dabit](#). I am a Developer Advocate at [AWS Mobile](#) working with projects like [AppSync](#) and [Amplify](#), and the founder of [React Native Training](#).

If you like React and React Native, checkout out our podcast — [React Native Radio](#) on [Devchat.tv](#).

Also, check out my book, [React Native in Action](#) now available from Manning Publications

If you enjoyed this article, please recommend and share it! Thanks for your time





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

