ReactJS - Animations

## Step 1 - Install React CSS Transitions Group

This is React add-on used for creating basic CSS transitions and animations. We will install it from the **command prompt** window −

C:\Users\username\Desktop\reactApp>npm install react-addons-css-transition-group

## Step 2 - Add a CSS file

Let's create a new folder **css** and file **style.css** inside. To be able to use it in the app, we need to link it to the **head** element in **index.html**.

<link rel = "stylesheet" type = "text/css" href = "css/style.css">

## Step 3 - Appear Animation

We will create a basic React component. The **ReactCSSTransitionGroup**element will be used as a wrapper of the component we want to animate. It will use **transitionAppear** and **transitionAppearTimeout**, while **transitionEnter** and **transitionLeave** are false.

### App.jsx

import React from 'react';

var ReactCSSTransitionGroup = require('react-addons-css-transition-group');

class App extends React.Component {

render() {

return (

<div>

<ReactCSSTransitionGroup transitionName = "example"

transitionAppear = {true} transitionAppearTimeout = {500}

transitionEnter = {false} transitionLeave = {false}>

<h1>My Element...</h1>

</ReactCSSTransitionGroup>

</div>

);

}

}

export default App;

### main.js

import React from 'react'

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));

The CSS animation is very simple.

### css/style.css

.example-appear {

opacity: 0.01;

}

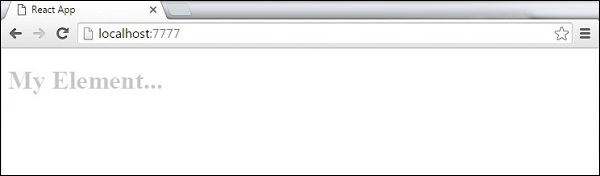
.example-appear.example-appear-active {

opacity: 1;

transition: opacity 500ms ease-in;

}

Once we start the app, the element will fade in.



## Step 4 - Enter and Leave Animations

Enter and leave animations can be used when we want to add or remove elements from the list.

### App.jsx

import React from 'react';

var ReactCSSTransitionGroup = require('react-addons-css-transition-group');

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

items: ['Item 1...', 'Item 2...', 'Item 3...', 'Item 4...']

}

this.handleAdd = this.handleAdd.bind(this);

};

handleAdd() {

var newItems = this.state.items.concat([prompt('Create New Item')]);

this.setState({items: newItems});

}

handleRemove(i) {

var newItems = this.state.items.slice();

newItems.splice(i, 1);

this.setState({items: newItems});

}

render() {

var items = this.state.items.map(function(item, i) {

return (

<div key = {item} onClick = {this.handleRemove.bind(this, i)}>

{item}

</div>

);

}.bind(this));

return (

<div>

<button onClick = {this.handleAdd}>Add Item</button>

<ReactCSSTransitionGroup transitionName = "example"

transitionEnterTimeout = {500} transitionLeaveTimeout = {500}>

{items}

</ReactCSSTransitionGroup>

</div>

);

}

}

export default App;

### main.js

import React from 'react'

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));

### css/style.css

.example-enter {

opacity: 0.01;

}

.example-enter.example-enter-active {

opacity: 1;

transition: opacity 500ms ease-in;

}

.example-leave {

opacity: 1;

}

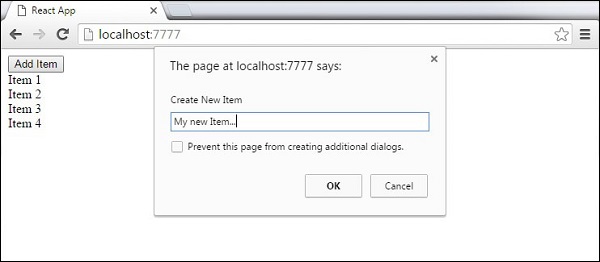
.example-leave.example-leave-active {

opacity: 0.01;

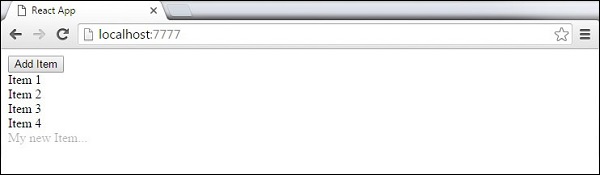
transition: opacity 500ms ease-in;

}

When we start the app and click the **Add Item** button, the prompt will appear.



Once we enter the name and press OK, the new element will fade in.



Now we can delete some of the items (**Item 3...**) by clicking it. This item will fade out from the list.



# Add CSS Class

var styles = {

transparentBg: {

background: 'transparent'

}

};

For example, you may have a CSS file like this:

.btn {

border: 1px solid black;

border-radius: 2px;

}

.icon {

background: url(...);

}

import React from 'react';

import s from './Button.css';

function Button() {

return (

<button className={s.btn}><i className={s.icon} />Click Me</button>

);

}

export default Button;

This configures every JavaScript file to be run through the react-hot loader, which configures hot module loading, and babel, which will transpire ES2015 features and the JSX syntax.

What we need to do is add another configuration for .css files where we first configure style-loader, and then css-loader:

{

test: /\.css$/,

loader: 'style-loader'

}, {

test: /\.css$/,

loader: 'css-loader',

query: {

modules: true,

localIdentName: '[name]\_\_[local]\_\_\_[hash:base64:5]'

}

}

# Install Bootstrap

$ npm install --save react react-dom

$ npm install --save react-bootstrap

## Stylesheets

Because React-Bootstrap doesn't depend on a very precise version of Bootstrap, we don't ship with any included css. However, some stylesheet **is required** to use these components. How and which bootstrap styles you include is up to you, but the simplest way is to include the latest styles from the CDN.

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap.min.css">

<!-- Optional theme -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap-theme.min.css">

# ReactJS - Higher Order Components

Higher order components are JavaScript functions used for adding additional functionalities to the existing component. These functions are **pure**, which means they are receiving data and returning values according to that data. If the data changes, higher order functions are re-run with different data input. If we want to update our returning component, we don't have to change the HOC. All we need to do is change the data that our function is using.

**Higher Order Component** (HOC) is wrapping around "normal" component and provide additional data input. It is actually a function that takes one component and returns another component that wraps the original one.

Let us take a look at a simple example to easily understand how this concept works. The **MyHOC** is a higher order function that is used only to pass data to **MyComponent**. This function takes **MyComponent**, enhances it with **newData** and returns the enhanced component that will be rendered on the screen.

import React from 'react';

var newData = {

data: 'Data from HOC...',

}

var MyHOC = ComposedComponent ⇒ class extends React.Component {

componentDidMount() {

this.setState({

data: newData.data

});

}

render() {

return <ComposedComponent {...this.props} {...this.state} />;

}

};

class MyComponent extends React.Component {

render() {

return (

<div>

<h1>{this.props.data}</h1>

</div>

)

}

}

export default MyHOC(MyComponent);

If we run the app, we will see that data is passed to **MyComponent**.



**Note** − Higher order components can be used for different functionalities. These pure functions are the essence of functional programming. Once you are used to it, you will notice how your app is becoming easier to maintain or to upgrade.