ReactJS Tutorial

Top of Form

 PDF Version [Quick Guide](https://www.tutorialspoint.com/reactjs/reactjs_quick_guide.htm) [Resources](https://www.tutorialspoint.com/reactjs/reactjs_useful_resources.htm)  Job Search [Discussion](https://www.tutorialspoint.com/reactjs/reactjs_discussion.htm)

Bottom of Form

**React** is front end library developed by Facebook. It's used for handling **view** layer for web and mobile apps. ReactJS allows us to create reusable UI components. It is currently one of the most popular JavaScript libraries and it has strong foundation and large community behind it.

Audience

This tutorial will help JavaScript developers that are diving in the ReactJS world for the first time. We will try to introduce every concept by showing simple code examples that can be easily understood. After finishing all the chapters, you will feel confident working with react. As a bonus we will introduce additional elements that work nice with ReactJS to help you learn best practices and follow the modern JavaScript trends.

Prerequisites

If you want to work with ReactJS, you need to have solid knowledge of **JavaScript**, **HTML5** and **CSS**. Even though ReactJS doesn't use HTML, the JSX is similar so your HTML knowledge will be very helpful. We will explain this more in one of the next chapters. We will also use **EcmaScript 2015** syntax so any knowledge in this area can be helpful.

# ReactJS - Overview

Advertisements

[Previous Page](https://www.tutorialspoint.com/reactjs/index.htm)

[Next Page](https://www.tutorialspoint.com/reactjs/reactjs_environment_setup.htm)

We already stated that ReactJS is JavaScript library used for building reusable UI components. Definition can be found on React official documentation −

React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, giving a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow which reduces boilerplate and is easier to reason about than traditional data binding.

## React Features

* **JSX** − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
* **Components** − React is all about components. You need to think of everything as a component. This will help you to maintain the code when working on larger scale projects.
* **Unidirectional data flow and Flux** − React implements one way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
* **License** − React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

## React Advantages

* React uses virtual DOM which is JavaScript object. This will improve apps performance since JavaScript virtual DOM is faster than the regular DOM.
* React can be used on client and server side.
* Component and Data patterns improve readability which helps to maintain larger apps.
* React can be used with other frameworks.

## React Limitations

* React only covers view layer of the app so you still need to choose other technologies to get a complete tooling set for development.
* React is using inline templating and JSX. This can seem awkward to some developers.

# ReactJS - Environment Setup

In this tutorial we will show you how to set up environment for successful React development. Notice that there are many steps involved but this will help you to speed up development process later. We will need **NodeJS** so if you don't have it installed, check the link from the table below.

|  |  |  |
| --- | --- | --- |
| **SN** | **Software** | **Description** |
| 1 | NodeJS and NPM | NodeJS is the platform needed for the Cordova development. Checkout our **[NodeJS Environment Setup](https://www.tutorialspoint.com/nodejs/nodejs_environment_setup.htm" \t "_blank)**. |

## Step 1 - Install Global Packages

You will need to install several packages for this setup. We will need some of the **babel** plugins so let's first install **babel** by running the following code in **command prompt** window.

C:\Users\username>npm install -g babel

C:\Users\username>npm install -g babel-cli

## Step 2 - Create Root Folder

The root folder will be named **reactApp** and we will place it on **Desktop**. After the folder is created we need to open it and create empty **package.json** file inside by running **npm init** from the **command prompt** and follow the instructions.

C:\Users\username\Desktop>mkdir reactApp

C:\Users\username\Desktop\reactApp>npm init

## Step 3 - Add Dependencies and plugins

We will use **webpack** bundler in these tutorials so let's install **webpack** and **webpack-dev-server**.

C:\Users\username>npm install webpack --save

C:\Users\username>npm install webpack-dev-server --save

Since we want to use React, we need to install it first. The **--save** command will add these packages to **package.json** file.

C:\Users\username\Desktop\reactApp>npm install react --save

C:\Users\username\Desktop\reactApp>npm install react-dom --save

We already mentioned that we will need some **babel** plugins so let's install it too.

C:\Users\username\Desktop\reactApp>npm install babel-core

C:\Users\username\Desktop\reactApp>npm install babel-loader

C:\Users\username\Desktop\reactApp>npm install babel-preset-react

C:\Users\username\Desktop\reactApp>npm install babel-preset-es2015

## Step 4 - Create files

Let's create several files that we need. You can add it manually or you can use **command prompt**.

C:\Users\username\Desktop\reactApp>touch index.html

C:\Users\username\Desktop\reactApp>touch App.jsx

C:\Users\username\Desktop\reactApp>touch main.js

C:\Users\username\Desktop\reactApp>touch webpack.config.js

## Step 5 - Set Compiler, Server and Loaders

Open **webpack-config.js** file and add the code below. We are setting webpack entry point to be **main.js**. Output path is the place where bundled app will be served. We are also setting development server to **8080** port. You can choose any port you want. And lastly, we are setting babel loaders to search for **js** files and use **es2015** and **react** presets that we installed before.

### webpack.config.js

var config = {

entry: './main.js',

output: {

path:'./',

filename: 'index.js',

},

devServer: {

inline: true,

port: 8080

},

module: {

loaders: [

{

test: /\.jsx?$/,

exclude: /node\_modules/,

loader: 'babel',

query: {

presets: ['es2015', 'react']

}

}

]

}

}

module.exports = config;

Open the **package.json** and delete **"test" "echo \"Error: no test specified\" && exit 1"** inside **"scripts"** object. We are deleting this line since we will not do any testing in this tutorials. Let's add the **start** command instead.

"start": "webpack-dev-server --hot"

Now we can use **npm start** command to start the server. **--hot** command will add live reload after something is changed inside our files so we don't need to refresh the browser every time we change our code.

## Step 6 - index.html

This is just regular HTML. We are setting **div id = "app"** as a root element for our app and adding **index.js** script which is our bundled app file.

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<title>React App</title>

</head>

<body>

<div id = "app"></div>

<script src = "index.js"></script>

</body>

</html>

## Step 7 - App.jsx and main.js

This is the first react component. We will explain React components in depth in one of our later tutorials. This component will render **Hello World!!!**.

### App.jsx

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

Hello World!!!

</div>

);

}

}

export default App;

We need to import this component and render it to our root **App** element so we can see it in browser.

### main.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

### NOTE

Whenever you want to use something, you need to **import** it first. If you want to make component usable in other parts of the app, you need to **export** it after creation and **import** it in the file where you want to use it.

## Step 8 - Running the Server

The setup is finished and we can start the server by running:

C:\Users\username\Desktop\reactApp>npm start

It will show you the port we need to open in browser, in our case **http://localhost:8080/**. After we open it we will see the following output:

# ReactJS - JSX

Advertisements

[Previous Page](https://www.tutorialspoint.com/reactjs/reactjs_environment_setup.htm)

[Next Page](https://www.tutorialspoint.com/reactjs/reactjs_components.htm)

React uses JSX for templating instead of regular JavaScript. It is not necessary to use it, but there are some pros that comes with it.

* JSX is faster because it performs optimization while compiling code to JavaScript.
* It is also type-safe and most of the errors can be caught during compilation.
* JSX makes it easier and faster to write templates if you are familiar with HTML.

## Using JSX

JSX looks like regular HTML in most cases. We already used it in environment setup tutorial. Look at the code from **App.jsx** where we are returning **div**.

### App.jsx

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

Hello World!!!

</div>

);

}

}

export default App;



Even though it's similar to HTML, there are a couple of things you need to keep in mind when working with JSX.

## Nested Elements

If you want to return more elements, you need to wrap it with one container element. Notice how we are using **div** as a wrapper for **h1**, **h2** and **p** elements.

### App.jsx

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

<h2>Content</h2>

<p>This is the content!!!</p>

</div>

);

}

}

export default App;



## Attributes

You can use your own custom attributes in addition to regular HTML properties and attributes. When you want to add custom attribute, you need to use **data-** prefix. In example below we added **data-myattribute** as an attribute of **p** element.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

<h2>Content</h2>

<p data-myattribute = "somevalue">This is the content!!!</p>

</div>

);

}

}

export default App;

## JavaScript Expressions

JavaScript expressions can be used inside of JSX. You just need to wrap it with curly brackets **{}**. Example below will render **2**.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>{1+1}</h1>

</div>

);

}

}

export default App;



You can not use **if else** statements inside JSX but you can use **conditional (ternary)** expressions instead. In example below variable **i** equals to **1** so the browser will render **true**, if we change it to some other value it will render **false**.

import React from 'react';

class App extends React.Component {

render() {

var i = 1;

return (

<div>

<h1>{i == 1 ? 'True!' : 'False'}</h1>

</div>

);

}

}

export default App;



## Styling

React recommends using inline styles. When you want to set inline styles, you need to use **camelCase** syntax. React will also automatically append **px** after the number value on specific elements. You can see below how to add **myStyle** inline to **h1** element.

import React from 'react';

class App extends React.Component {

render() {

var myStyle = {

fontSize: 100,

color: '#FF0000'

}

return (

<div>

<h1 style = {myStyle}>Header</h1>

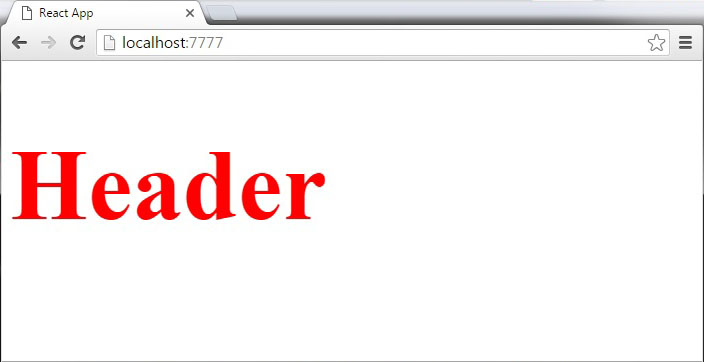
</div>

);

}

}

export default App;



## Comments

When writing comments you need to put curly brackets **{}** when you want to write comment within children section of a tag. It is good practice to always use **{}** when writing comments since you want to be consistent when writing the app.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

{//End of the line Comment...}

{/\*Multi line comment...\*/}

</div>

);

}

}

export default App;

## Naming Convention

HTML tags are always using **lowercase** tag names, while React components starts with **Uppercase**.

### NOTE

You should use **className** and **htmlFor** as XML attribute names instead of **class** and **for**.

This is explained on React official page −

Since JSX is JavaScript, identifiers such as **class** and **for** are discouraged as XML attribute names. Instead, React DOM components expect DOM property names like **className** and **htmlFor**, respectively.