**React**

ReactJS is JavaScript front-end library used for building reusable UI , developed by Facebook components.

It is used for handling the view layer for web and mobile apps. ReactJS allows us to create reusable UI components.

## React eatures

* **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 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

### **How is React different**

*Because React is a small library focused on building UI components, it is necessarily different than a lot of other JavaScript frameworks.*

*For example, AngularJS (1.x) approaches building an application by extending HTML markup and injecting various constructs (e.g. Directives, Controllers, Services) at runtime. As a result, AngularJS is very opinionated about the greater architecture of your application — these abstractions are certainly useful in some cases, but in many situations, they come at the cost of flexibility.*

*By contrast, React focuses exclusively on the creation of components, and has few (if any) opinions about an application’s architecture. This allows a developer an incredible amount of flexibility in choosing the architecture they deem “best” — though it also places the responsibility of choosing (or building) those parts on the developer.*

## Using JSX

JSX looks like a regular HTML in most cases. We already used it in the Environment Setup chapter. Look at the code from **App.jsx** where we are returning **div**.

When Facebook first released React to the world, they also introduced a new dialect of JavaScript called JSX that embeds raw HTML templates inside JavaScript code. JSX code by itself cannot be read by the browser; it must be transpiled into traditional JavaScript using tools like Babel and webpack. While many developers understandably have initial knee-jerk reactions against it, JSX (in tandem with ES2015) has become the defacto method of defining React components.

### **App.jsx**

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

Hello World!!!

</div>

);

}

}

export default App;

We cannot use **if else** statements inside JSX instead we can use **conditional (ternary)** expressions

Comments

When writing comments, we need to put curly brackets **{}** when we want to write comment within children section of a tag. It is a good practice to always use **{}** when writing comments, since we 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>

);

}

}

# **ReactJS - Environment Setup**

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

|  |  |
| --- | --- |
| **Sr. No.** | **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). |

## Step 1 - Create the 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 2 - Install Global Packages

We 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 the **command prompt** window.

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

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

## Step 3 - Add Dependencies and Plugins

We will use **webpack** bundler in these tutorial. Let's install **webpack** and **webpack-dev-server**.

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

C:\Users\username\Desktop\reactApp>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

As already mentioned, 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 the Files

Let's create several files that we need. It can be added manually or using the **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

Alternative way to create files that we need

C:\Users\username\Desktop\reactApp>type nul >index.html

C:\Users\username\Desktop\reactApp>type nul >App.jsx

C:\Users\username\Desktop\reactApp>type nul >main.js

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

## Step 5 - Set Compiler, Server and Loaders

Open **webpack.config.js** file and add the following code. 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 the 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-loader',

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 tutorial. Let's add the **start** command instead.

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

Before the above step, it will required **webpack-dev-server**. To install **webpack-dev-server**, use the following command.

C:\Users\username\Desktop\reactApp>npm install webpack-dev-server -g

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 a subsequent chapter. 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 the 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 the 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 complete and we can start the server by running the following command.

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

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



|  |  |  |
| --- | --- | --- |
| **Technology** | **AngularJS** | **React** |
| Developer | Google | Facebook |
| Technology type | Full-fledged MVC framework written in JavaScript | JavaScript library (View in MVC; requires Flux to implement architecture) |
| Concept | Brings JavaScript into HTML Works with the real DOM Client-side rendering | Brings HTML into JavaScript Works with the virtual DOM Server-side rendering |
| Data binding | Two-way data binding | One-way data binding |
| Dependencies | Manages dependencies automatically | Requires additional tools to manage dependencies |
| Language | JavaScript + HTML | JavaScript + JSX |
| Last version | AngularJS 1.6.0 RC2 | React 15 |
| Suits best | Best for SPAs that update single view at a time | Best for SPAs that update many views at a time |