I followed these tutorials:

<https://www.youtube.com/watch?v=g6a5yC_v70c&list=PL6gx4Cwl9DGBuKtLgPR_zWYnrwv-JllpA&index=8>

To implement ReactJS add these 3 scripts in your view:

<**script src="scripts/react.15.3.0.min.js"**></**script**>  
<**script src="scripts/react-dom.15.3.0.min.js"**></**script**>  
*<!-- Babel - JSX Transpiler -->*<**script src="scripts/browser.5.8.24.min.js"**></**script**>

**react.js**

Contains core ReactJS framework.

**react-dom.js**

Contains React API to modify DOM.

**browser.js**

Is a Babel JSX Transpiler. Using JSX we can combine HTML and JavaScript.

**NOTE:**

To get detailed error message don’t user minified react.js and react-dom.js

**Bootstrap ReactJS**

Create a container using <**div id="reactContainer"**></**div**> and add Components to it using **ReactDOM**.render(<**h2**>React Heading</**h2**>, **document**.getElementById(**"reactContainer"**));.

Anything written inside <**script type="text/babel"**></**script**> will be transpiled by Babel. And we can write JSX code.

**Example:**

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**script src="scripts/react.15.3.0.min.js"**></**script**>  
 <**script src="scripts/react-dom.15.3.0.min.js"**></**script**>  
 *<!-- Babel - JSX Transpiler -->* <**script src="scripts/browser.5.8.24.min.js"**></**script**>  
 <**title**>React JS</**title**>  
 <**style**>  
 **\*** {**margin**: 0;**padding**: 0;**font-family**: **Arial**;}  
 </**style**>  
</**head**>  
<**body**>  
  
<**div id="reactContainer"**></**div**>  
  
*<!-- Any thing defied in "text/babel" will be transpiled by babel (https://babeljs.io/) script browser.5.8.24.min.js -->  
<!-- By Doing this we can write JSX code -->*<**script type="text/babel"**>  
 **ReactDOM**.render(<**h2**>React Heading</**h2**>, **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>  
</**html**>

**Creating Custom Component**

Everything in React JS is a Component. Just like everything in HTML is an Element.

We can create our own custom components in React JS and render it on DOM.

**IMPORTANT!**

React components name must begin with capital letter.

This is how JSX differentiate React components and HTML elements.

<https://facebook.github.io/react/docs/jsx-in-depth.html#html-tags-vs.-react-components>

Any React component that needs UI. Should contain render function. This render Function return html that it needs to render.

**Example:**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.**createClass**({  
render: **function** () {  
 **return** (<**h1**>My Custom Component</**h1**>);  
 }  
 });  
 **ReactDOM**.render(<**MyCustomComponent**/>, **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>

**Single Root XML Node**

When writing JSX in custom component's return function OR in ReactDOM.render() function, then it could only have Single XML Root Node. And It has be a well defined XML just like XHTML.

**NOTE:**

I used <div> tag to enclose multiple tags tags in MyCustomComponent class definition and I used <div> tag to enclose multiple tags tags in ReactDOM.render().

Just to have single Root Node

**NOTE:**

I closed <input placeholder="Input"></input> tag because JSX has to be well formed XML.

**Example:**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div**>  
 <**label**>Input:</**label**>  
 <**input placeholder="Input"**></**input**>  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.render((  
 <**div**>  
 <**MyCustomComponent**/>  
 <**MyCustomComponent**/>  
 <**MyCustomComponent**/>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>

**Accessing Custom Component’s Attribute Values**

Any custom arguments we send to component can be accessed through "this.props" object inside component.

**NOTE:**

In JSX cannot use this.props in quotes.

<http://stackoverflow.com/questions/21668025/react-jsx-access-props-in-quotes>

**NOTE:**

In JSX + Harmony OR ES6 we can use backtick (`) to combine static text and this.props

**NOTE:**

How I used placeholder={this.props.placeholder}

In the rendered page HTML placeholder's value is enclosed in double quotes. Not sure if browser did it or JSX did it.

**Example**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div**>  
 <**label**>{**this**.**props**.**label**}:</**label**>  
 <**input placeholder=**{**this**.**props**.**placeholder**}></**input**>  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.render((  
 <**div**>  
 <**p**>  
 <**MyCustomComponent label="Name" placeholder="Please enter Name..."**/>  
 </**p**>  
 <**p**>  
 <**MyCustomComponent label="Address" placeholder="Please enter Address..."**/>  
 </**p**>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>

**Accessing Custom Component’s Children Elements**

Elements within component can be accessed using {this.props.children} object.

**NOTE:**

In JSX code we used "className=" instead of "class=". This is because "class" is a reserved ES5 or ES6 keyword.

**Example**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.createClass({  
 render: **function** () {  
 **return** (  
 <**div className="greenBox"**>  
 <**h1**>My Custom Component</**h1**>  
 {**this**.**props**.children}  
 Some Detailed Text  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.render((  
 <**div**>  
 <**div className="blueBox"**>  
 <**MyCustomComponent**>  
 <**h2**>  
 My Component 1  
 </**h2**>  
 </**MyCustomComponent**>  
 </**div**>  
 <**div className="blueBox"**>  
 <**MyCustomComponent**>  
 <**h2**>  
 My Component 2  
 </**h2**>  
 </**MyCustomComponent**>  
 </**div**>  
 <**div className="blueBox"**>  
 <**MyCustomComponent**>  
 <**h2**>  
 My Component 3  
 </**h2**>  
 </**MyCustomComponent**>  
 </**div**>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>

**Event Handling / Calling functions**

**NOTE:**

We used (onClick=””) instead of (onclick=””).

**Example**

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**script src="scripts/react.15.3.0.min.js"**></**script**>  
 <**script src="scripts/react-dom.15.3.0.min.js"**></**script**>  
 *<!-- Babel - JSX Transpiler -->* <**script src="scripts/browser.5.8.24.min.js"**></**script**>  
 <**title**>React JS</**title**>  
 <**style**>  
 **\*** {**margin**: 0;**padding**: 0;**font-family**: **Arial**;}  
  
 .**blueBox**, .**greenBox**, .**redBox**, .**darkBlueBox**, .**whiteBox** {  
 **padding**: 20**px**;  
 **color**: **white**;  
 **font-size**: 20**px**;  
 **border**: 1**px solid #717171**;  
 **text-decoration**: **none**;  
 **margin**: 10**px**;  
 }  
  
 .**blueBox** {  
 **background-color**: **#8cbeed**;  
 }  
  
 .**whiteBox** {  
 **background-color**: **#fcfcfc**;  
 **color**: **#3d3d3d**;  
 }  
  
 .**greenBox** {  
 **background-color**: **#9ed992**;  
 }  
  
 .**darkBlueBox** {  
 **background-color**: **#2c4a80**;  
 }  
  
 .**redBox** {  
 **background-color**: **#ad3a26**;  
 }  
  
 .**button** {  
 **box-shadow**: 0 0 3**px rgba**(0, 0, 0, 0.70);  
 }  
  
 .**button**:**hover** {  
 **box-shadow**: 3**px** 3**px** 5**px rgba**(0, 0, 0, 0.70);  
 }  
 .**button**:**active** {  
 **box-shadow**: **inset** 0 0 0 9999**px rgba**(0, 0, 0, 0.2);  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
 **function** *myFunction*() {  
 alert(**"Function 2"**);  
 }  
  
 **var *MyCustomComponent*** = **React**.createClass({  
 myFunction1: **function** () {  
 alert(**"Function 1"**);  
 },  
 myFunction2: **function** () {  
 *myFunction*();  
 },  
 render: **function** () {  
 **return** (  
 <**div className="redBox"**>  
 <**div**>  
 {**this**.**props**.children}  
 </**div**>  
 <**button onClick=**{**this**.myFunction1} **className="darkBlueBox button"**>Button 1</**button**>  
 <**button onClick=**{**this**.myFunction2} **className="whiteBox button"**>Button 2</**button**>  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.render((  
 <**div**>  
 <**MyCustomComponent**>Component 1</**MyCustomComponent**>  
 <**MyCustomComponent**>Component 2</**MyCustomComponent**>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>  
</**html**>

**State**

Every React component have an object called this.state. Just like component's attribute we could use STATE to customize a component.

Whenever a component need a value that could change and have an effect during life of component we use STATE.

Whenever a component need a value would stay same through its life then we use prop/attribute.

States are watched by React. Means that if a state value is changed then it will impact anywhere that the state values is used in render UI.

IMPORTANT:

If the state changes then render function runs again. That in-turn will re-render all its children components. We could control it using shouldComponentUpdate property

<https://facebook.github.io/react/docs/component-specs.html#updating-shouldcomponentupdate>

<http://stackoverflow.com/questions/24718709/reactjs-does-render-get-called-any-time-setstate-is-called>

**NOTE:**

getInitialState: just like "render" function, getInitialState will be called by ReactJS.

Returned valued from the will be the initial state. A component could only have single state but

could have any value. To store multiple values in the state, we could have it return an object.

e.g. Note how it return curly braces {}

<https://facebook.github.io/react/docs/component-specs.html#getinitialstate>

<https://facebook.github.io/react/tips/props-in-getInitialState-as-anti-pattern.html>

**NOTE:**

Checkbox have a property called defaultChecked. Its NOT a special React or JSX property.

<http://www.w3schools.com/jsref/prop_checkbox_defaultchecked.asp>

**Example:**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.**createClass**({  
 getInitialState: **function** () {  
 **return** {**checked**: **true**};  
 },  
 handleChecked: **function** () {  
 **this**.setState({**checked**: !**this**.**state**.**checked**});  
 },  
 render: **function** () {  
 **var** message;  
 **if** (**this**.**state**.**checked**) {  
 message = **"checked"** } **else** {  
 message = **"unchecked"** }  
  
 **return** (  
 <**div className="greenBox"**>  
 <**input className="blueBox button" type="checkbox" onChange=**{**this**.handleChecked} **defaultChecked=**{**this**.**state**.**checked**}/>  
 <**div className="whiteBox"**>Check box is {message}.</**div**>  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.**render**(<**MyCustomComponent**/>, **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>

**Refs**

Refs could be used just like we use document.getElementById("Element\_ID").

In React Components we don't use element id to get references to an element instead we use this.refs.egElementName

Every React component will have its own private this.refs object. In this object it will list of elements that have been give ref="" property/attribute.

**Example**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
**var *MyCustomComponent*** = **React**.**createClass**({  
 showText: **function** () {  
 alert(**this**.**refs**.myTextArea.**value**);  
 },  
 render: **function** () {  
 **return** (  
 <**div className="blueBox"**>  
 Enter Comments:  
 <**div**>  
 <**textarea ref="myTextArea" className="whiteBox"**>  
 {**this**.**props**.**children**}  
 </**textarea**>  
 </**div**>  
 <**button onClick=**{**this**.showText} **className="redBox button"**>Show Text</**button**>  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.**render**((  
 <**div**>  
 <**MyCustomComponent**>Comments 1</**MyCustomComponent**>  
 <**MyCustomComponent**>Comments 2</**MyCustomComponent**>  
 <**MyCustomComponent**>Comments 3</**MyCustomComponent**>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>

**Nesting Components & Passing Properties from Parent Component to Child Component**

**Example**

<**body**>  
<**div id="reactContainer"**></**div**>  
<**script type="text/babel"**>  
 **var *ParentComponent*** = **React**.**createClass**({  
 render: **function** () {  
 **var** child = **"c "** + **this**.**props**.parentName;  
 **return** (  
 <**div className="redBox"**>  
 Parent = {**this**.**props**.parentName}  
 <**ChildComponent childName=**{child}/>  
 </**div**>  
 );  
 }  
 });  
  
 **var *ChildComponent*** = **React**.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div className="whiteBox"**>  
 Child = {**this**.**props**.childName}  
 </**div**>  
 );  
 }  
 });  
  
 **ReactDOM**.**render**((  
 <**div**>  
 <**ParentComponent parentName="1"**/>  
 <**ParentComponent parentName="2"**/>  
 <**ParentComponent parentName="3"**/>  
 </**div**>  
 ), **document**.getElementById(**"reactContainer"**));  
</**script**>  
</**body**>