# React JS

**Front-end development** refers to what the end user (also commonly referred to as the "**client**") can see. In the most basic forms, front-end development consists of HTML, CSS, and JavaScript.

Eventually, developers decided that there must be a better way to manage all of that code, so they created libraries that could make life easier. **React**was one of those libraries.

React was created by Facebook and released to the public in May of 2013 and has been consistently maintained since then.

* React is used to build user interfaces (UI) on the front end.
* React is the **view** layer of an MVC application (Model View Controller)

## React - Introduction

**React** is one of the most popular JavaScript libraries for **front-end web applications**.  
  
Interactive websites need to update the DOM (Document Object Model) each time a change happens. This process is generally resourceful and slow.

Compared to other libraries that manipulate the DOM, React uses a **Virtual DOM**, allowing to update only the parts of the website that have changed. This increases the speed of updates dramatically, as modern web applications may contain thousands of elements.

#### Installation

To work with React, we don’t need to install anything. We just need to add following js files.

* [**React**](https://reactjs.org/docs/react-api.html) - the React top level API
* [**React DOM**](https://reactjs.org/docs/react-dom.html) - adds DOM-specific methods
* [**Babel**](https://babeljs.io/) - a JavaScript compiler that lets us use ES6+ in old browsers

First, we need to add the React library as two **script** tags to the **head** of our HTML document:

<script src="https://unpkg.com/react@16/umd/**react.development.js**" crossorigin></script>   
<script src="https://unpkg.com/react-dom@16/umd/**react-dom.development.js**" crossorigin />

To enable the use of **JSX, use babel**. **JSX** is a syntax extension to JavaScript.

<script src="https://unpkg.com/babel-standalone@6/**babel.min.js**"></script>

#### Hello, World – Using index.tml

After adding the required script tags, we can start building our React app!  
  
We need to add a container to display something using React.

<div id="container"></div>

ReactDOM.render(data, element)

<!DOCTYPE html>

<html>

    <head>

        <title>Test</title>

        <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin/>

        <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

        <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

    </head>

    <body>

        <div id="container"></div>

        <script type="text/babel">

        ReactDOM.render(

          <h1>Hello, React!</h1>,

          document.getElementById('container')

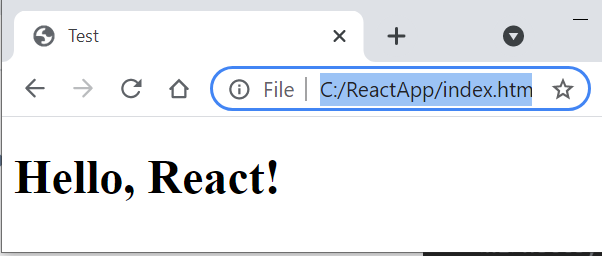
        )

        </script>

    </body>

</html>

The code finds the **container** <div> and adds the **h1** heading to it.



#### HelloWorld – Using Class

<!DOCTYPE html>

<html>

  <head>

    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>

    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>

  </head>

  <body>

    <div id="root"></div>

    <script type="text/babel">

      class App extends React.Component {

        render() {

          return <h1>Hello Satya!</h1>

        }

      }

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

    </script>

  </body>

</html>

In above we are return **single line** content, if we have **multiple lines** – we need to use one of below.

**1.using ( ) brackets**

    render() {

        return (

        <div className="App">

            <h1>Hello, React!</h1>

         </div>

        );

      }

**2.using [ ] – as Array of elements**

      render() {

        return [

          <ChildA key="key1" />,

          <ChildB key="key2" />,

          <ChildC key="key3" />,

        ];

      }

## React - Create React App

In the previous lesson we learned how to add React to a simple HTML document using the script tags.  
However, real web apps have a different scale, contain multiple files, use 3rd party libraries, etc.

Facebook has created a handy tool called [create-react-app](https://create-react-app.dev/) that makes it easy to setup a React project with just a simple command. **Create React App** allows us to focus on the code, rather than installing and configuring different tools.

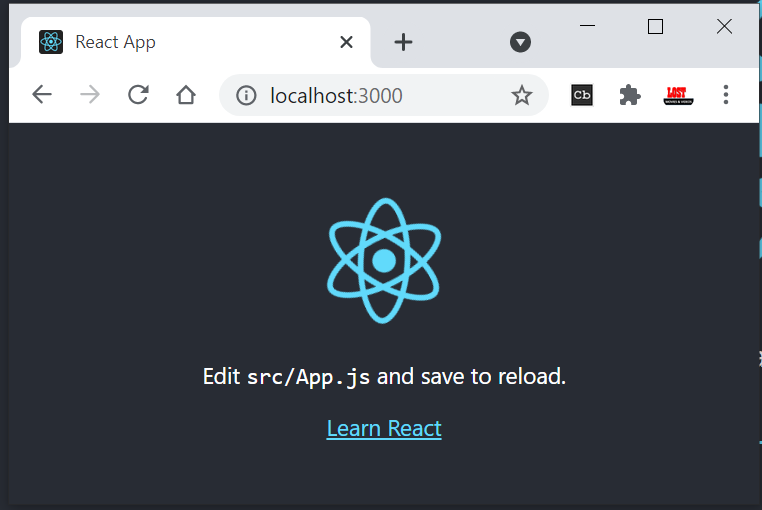
To get started, make sure you have a recent version of [**Node**](https://nodejs.org/en/) installed on your machine.  
Run the following commands in the Terminal to create and start a React app called "**ReactApp**":

npx create-react-app reactapp

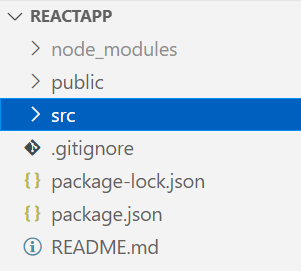
cd reactapp

npm start

This will install all the required dependencies, configure and start the project on **localhost:3000**.  
This is the default output of our project in the browser:



### Folder Structure



/public folder contains files related to how the application will display on the client, the most important of those being **index.html**, which is the HTML template of our page.

/src folder contains all of the JavaScript, CSS, and image files that will be compiled into a bundle file and injected into **index.html.**

The two files below are the only critical files:

* index.js: This file is the entry point into our application. In our code, a method called ReactDOM.render() is used to find an element with id="root" in the HTML and add content to it.
* app.js: This file is the main **component** that will be rendered to the DOM, which currently includes the React logo image and the default text, that we see in the output.

**Webpack** creates a "bundle" file containing the content of multiple files that need to be "bundled" together and it is all added together into a single file.

### Customizing React App

Go ahead and delete all the files out of the /src directory, and we'll create our own boilerplate file without any bloat. We'll just keep index.css and index.js.

Now in index.js, we're importing React, ReactDOM, and the CSS file.

import React from 'react'

import ReactDOM from 'react-dom'

import './index.css'

class App extends React.Component {

    render() {

        return (

        <div className="App">

            <h1>Hello, Customized App</h1>

         </div>

        );

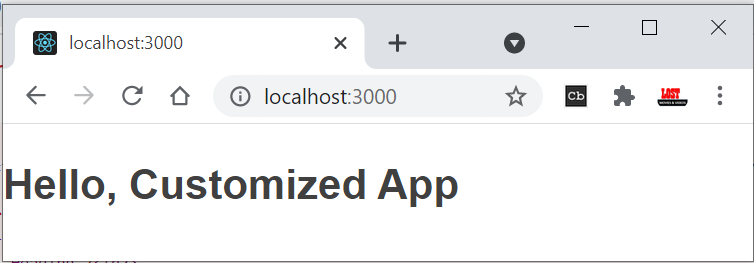
      }

}

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

In index.html, just place only <div> & start the Application

<div id="root"></div>



If we use import React, {Component} from 'react', No need to write complete extends class name

import React, {Component} from 'react'

class App extends Component {

}

## React - JSX - JavaScript + XML

ReactDOM.render(

**<h1>Hello, React!</h1>,**

document.getElementById('root')

);

We will start with the <h1>Hello, React!</h1> element.  
  
As you can see, the element is not in quotes to represent a string. It's like an HTML element, however we use it right in the JavaScript code!  
This is called **JSX**, and it is a syntax extension to JavaScript. It allows us to build UI elements right in the JavaScript code!

The code calls React's **render** method, and passes it two arguments, a **JSX element** and **a container**. The render method displays the provided element in the container, which, in our case, is the HTML element with id="root".

### Expressions in JSX

We can use any JavaScript expression inside JSX using curly braces.  
 **For example:**

const name = "David";

const el = <p>Hello, {name}</p>;

ReactDOM.render(

el,

document.getElementById('root')

);

### Attributes in JSX

We can specify attributes using quotes, just like in HTML:

<div **id="name"**></div>

When using a JavaScript expression as the attributes value, the quotes should not be used:

<div **id={user.id}**></div>

Finally, When the JSX expressions are compiled, they are converted into JavaScript **objects**, representing React elements.

React then uses these elements to build the corresponding HTML DOM and display it in the browser.

### Virtual DOM

We learned in the previous part that React updates only the elements that are necessary.  
This allows React apps to be much faster than apps built with other front-end technologies.  
  
**But how does React achieve that?**  
React uses a **Virtual DOM**, which is a lightweight representation of the DOM.  
When an element gets changed, it is first updated in the Virtual DOM. That process is fast, as the virtual DOM is represented by simple objects.  
  
After that, React compares the Virtual DOM to its previous state and only applies the DOM updates necessary to bring the DOM to the desired state.

## React – Components

So far, we've created one component - the App component. Most React apps have many small components, and everything loads into the main App component.

In React, there are two types of components that you can use:

* **Functional Components**
* **Class Components**.

### Functional Components

A functional component is a simple JavaScript function:

function Hello() {

  return <h1>Hello world.</h1>;

}

The code above defined a functional component called **Hello**, that returns a simple React element.

Notice that the name of the functional component begins with a **capital letter**. This is absolutely critical. If we start the name of a component with a lowercase letter, the browser will treat our component like a regular HTML element instead of a Component.

In order to display the component, we need to create the corresponding JSX element, by calling the function.  
  
For example, for our user-defined component **Hello**:

const el = <Hello/>;

Now, we can use our user-defined element and render it on the page:

function Hello() {

  return <h1>Hello world.</h1>;

}

const el = <Hello />;

ReactDOM.render(

  el,

  document.getElementById('root')

);

Remember, all component names need to start with a capital letter.

### Class Components

**Class components** are typically used when there are more advanced user interactions, like forms, and animations.  
  
All class components need to extend the React.Component class. Class components need to have a **render()**method, which is in charge of telling what the page should show.  
  
Remove the App class from index.js, place it in new file called app.js

//app.js

import React, {Component} from 'react'

class App extends Component {

    render() {

        return (

        <div className="App">

            <h1>Hello, Customized App</h1>

         </div>

        );

      }

}

export default App

Import in index.js

import React from 'react'

import ReactDOM from 'react-dom'

import './index.css'

import App from './app'

 //Code Removed

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

We export the component as App and load it in index.js.

In index.html, just place only <div> & start the Application

<div id="root"></div>

Example Simple Component

**simple component**, which is a function. This component doesn't use the class keyword. Let's take our **Table** and make two simple components for it - a **TableHeader**, and a **TableBody**.

We're going to use ES6 arrow functions to create these simple components.

First, the TableHeader

const TableHeader = () => {

    return (

      <thead>

        <tr>

          <th>Name</th>

          <th>Job</th>

        </tr>

      </thead>

    )

  }

Then the TableBody .

  const TableBody = () => {

    return (

      <tbody>

        <tr>

          <td>Charlie</td>

          <td>Janitor</td>

        </tr>

        <tr>

          <td>Mac</td>

          <td>Bouncer</td>

        </tr>

      </tbody>

    )

  }

Now our Table file will look like this. Note that the TableHeader and TableBody components are all in the same file and being used by the Table class component.

const TableHeader = () => { ... }

const TableBody = () => { ... }

class Table extends Component {

  render() {

    return (

      <table>

        <TableHeader />

        <TableBody />

      </table>

    )

  }

}

Example Class Component

We are created Table component to display table data in table.js – and export as Table component

import React, {Component} from 'react'

class Table extends Component {

  render() {

    return (

      <table>

        <thead>

          <tr>

            <th>Name</th>

            <th>Job</th>

          </tr>

        </thead>

        <tbody>

          <tr>

            <td>Charlie</td>

            <td>Janitor</td>

          </tr>

          <tr>

            <td>Mac</td>

            <td>Bouncer</td>

          </tr>

        </tbody>

      </table>

    )

  }

}

export default Table

Import Table in app.js – pass it to index.html

import React, {Component} from 'react'

import Table from './table'

class App extends Component {

  render() {

    return (

      <div className="container">

        <Table />

      </div>

    )

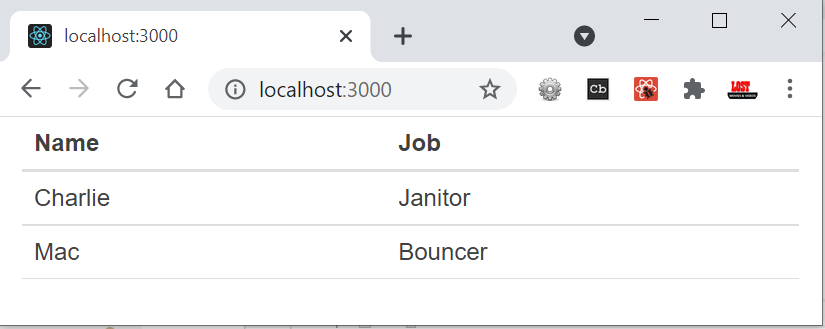
  }

}

export default App

In index.html, just place only <div> & start the Application

<div id="root"></div>



## React – Data Handling

To handle data in React, we do with properties, referred to as **props**, and with **state**

### Props

Functional components can accept arguments, similar to JavaScript functions. These arguments are called **props**, and represent an object.  
  
For example, we can use props in our Hello component:

function Hello(**props**) {

  return <p>Hello, {props.name}!</p>;

}

Props can be accessed in class components using this.props.

class Hello extends React.Component {

  render() {

    return <p>Hello, {this.props.name}!</p>;

  }

}

An important thing to consider is that props are **read-only**, meaning components cannot modify their props.

### State

Many web apps need their components to change their data, for example, after user interaction (clicking a button, submitting a form, etc.).However, **props** cannot be changed.  
  
In order to allow components to manage and change their data, React provides a feature called **state**.  
**State** is an object that is added as a property in class components.

**State** is just a simple object, that contains key:value pairs.Similar to props, the values can be accessed using **this.state**.

class Hello extends React.Component {

  state = {

    name: "James"

  }

  render() {

    return <h1>Hello {this.state.name}.</h1>;

  }

}

**Changing State**

**this.setState**({   
name: "James",   
age: 25   
});

## Lifecycle Methods

Enough, from now onwards read

<https://www.w3schools.com/react/react_props.asp>

## Ref.

<https://www.sololearn.com/learning/1097/3370/7610/2>

Best

<https://www.taniarascia.com/getting-started-with-react/>