Shivaprakash

ReactJS Tutorial – Design Your Web UI Using ReactJS JavaScript Library

Aug 2, 2017

Most of you would have heard about 'ReactJS' also known as React. For those of you curious to know more, I'll be covering all the core concepts of React you need to know. By the end of this ReactJS tutorial, I'm confident that you will be clear with all the fundamentals of React. Let me start by giving you an overview of what I'll be covering in this ReactJS tutorial.

- Evolution Of React
- Why Learn React?
- React Features Overview
- How does It work?
- Building Blocks
- React Installation

You may go through this recording of ReactJS Tutorial where our instructor has explained the topics in a detailed manner with examples that will help you to understand this concept better.

ReactJS Tutorial For Beginners | ReactJS Redux Training For Beginners | Edureka

Evolution Of React

React is a JavaScript library used to build the user interface for web applications. React was initially developed and maintained by the folks at Facebook, which was later used in their products (WhatsApp & Instagram). Now it is an open source project with an active developer community. Popular websites like Netflix, Airbnb, Yahoo!Mail, KhanAcademy, Dropbox and many more use React to build their UI. Modern

Evolution Of React

React is a JavaScript library used to build the user interface for web applications. React was initially developed and maintained by the folks at Facebook, which was later used in their products (WhatsApp & Instagram). Now it is an open source project with an active developer community. Popular websites like Netflix, Airbnb, Yahoo!Mail, KhanAcademy, Dropbox and many more use React to build their UI. Modern websites are built using MVC (model view controller) architecture. React is the 'V' in the MVC which stands for view, whereas the architecture is provided by **Redux** or **Flux**. React native is used to develop mobile apps, the Facebook mobile app is built using React native.

Facebook's annual F8 Developer conference 2017, saw two promising announcements: **React Fiber** and **ReactVR**. React Fiber is a complete rewrite of the previous release focusing on incremental rendering and quick responsiveness, React Fiber is backward compatible with all previous versions. ReactVR is built on top of React Native frameworks, it enables developing UI with the addition of 3D models to replicate 360-degree environment resulting in fully immersive VR content.

Why Learn React?

"Let's just write less and do more!!"

React is among the easiest JS libraries you can start with. Conventional Vanilla JavaScript is more time-consuming, why waste time writing lengthy code when u can get things done smoothly with React. React has over 71,200 stars on GitHub , making it the 4th most starred project of all time. After looking at the below example, I am sure you would understand why front-end developers across the world are switching to React. Now let's try coding a set of nested lists in React and compare it with conventional JavaScript syntax.

*Example: **30** lines of code in Vanilla JavaScript can be replaced by just **10** lines of React code, isn't that awesome!!*

React

```
<ol>

<li>List item 1 </li>


<li>List item 2 (child list)

<ul>

<li>Subitem 1</li>


<li>Subitem 2</li>

</ul>

</li>


<li>Final list item</li>

</ol>
```

Equivalent Vanilla JavaScript

```
React.createElement(
 "ol",
 null,
 React.createElement(
 "li",
 null,
 "List item 1 "
 ),
 React.createElement(
 "li",
 null,
 "List item 2 (child list)",
 React.createElement(
 "ul",
 null,
 "List item 1 "
 ),
 React.createElement(
 "li",
 null,
 "List item 2 (child list)",
 React.createElement(
 "ul",
 null,
 React.createElement(
 "li",
 null,
 "Subitem 1"
 ),
 React.createElement(
 "li",
 null,
 "Subitem 2"
 )
 )
 ),
 React.createElement(
 "li",
 null,
 "Final list item"
 )
);
```

As you have already figured it out when the complexity increases, the JavaScript code generated becomes unmanageable. This is where JSX comes to the rescue ensuring the code is short and easily readable.

ReactJS Tutorial- Key Terminology

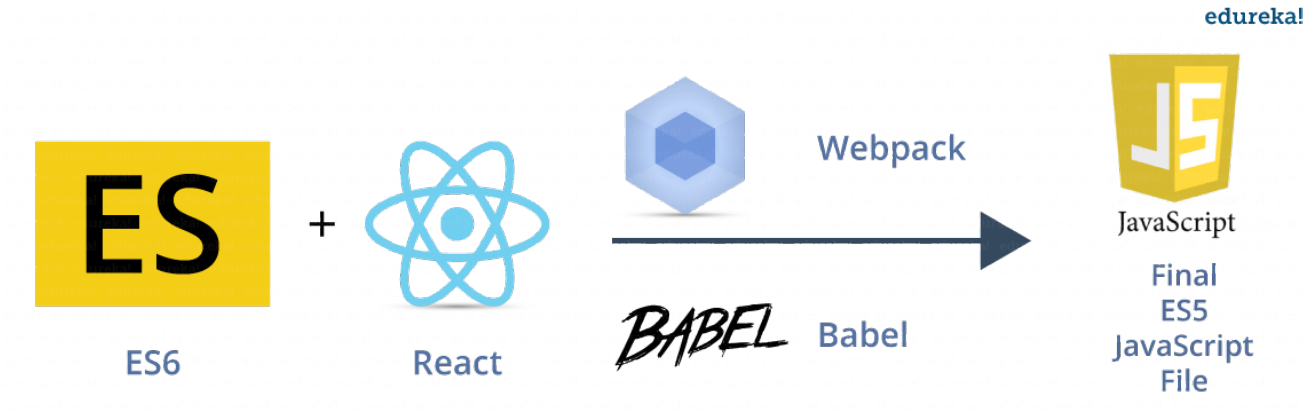


Figure: *ReactJS Tutorial – Dependencies*

Before we dive deeper into this ReactJS tutorial, let me first introduce you to some key terms you need to be familiar with.

JSX (JavaScript Extension)

JSX Allows us to include 'HTML' in the same file along with 'JavaScript' (HTML+JS=JSX). Each component in React generates some HTML which is rendered by the DOM.

ES6 (ES2015)

The sixth version of JavaScript is standardized by ECMA
International in 2015. Hence the language is referred to as ECMAScript. ES6 is not completely supported by all modern browsers.

ES5(ES2009)

This is the fifth JavaScript version and is widely accepted by all modern browsers, it is based on the 2009 ECMA specification standard. Tools are used to convert ES6 to ES5 during runtime.

Webpack

A module bundler which generates a build file joining all the dependencies.

Babel

This is the fifth JavaScript version and is widely accepted by all modern browsers; it is based on the 2009 ECMA specification standard. Tools are used to convert ES6 to ES5 during runtime.

Webpack

A module bundler which generates a build file joining all the dependencies.

Babel

This is the tool used to convert ES6 to ES5. This is done because not all web browsers can render React (ES6+JSX) directly.

GET STARTED WITH REACT TODAY

React Features Overview

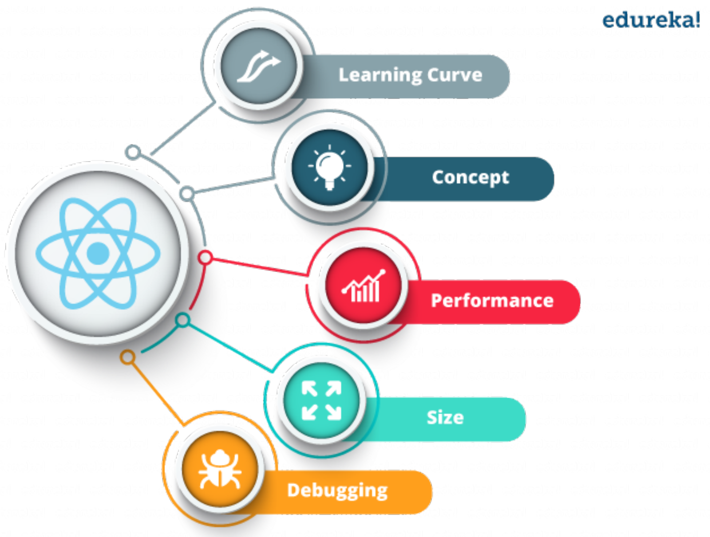


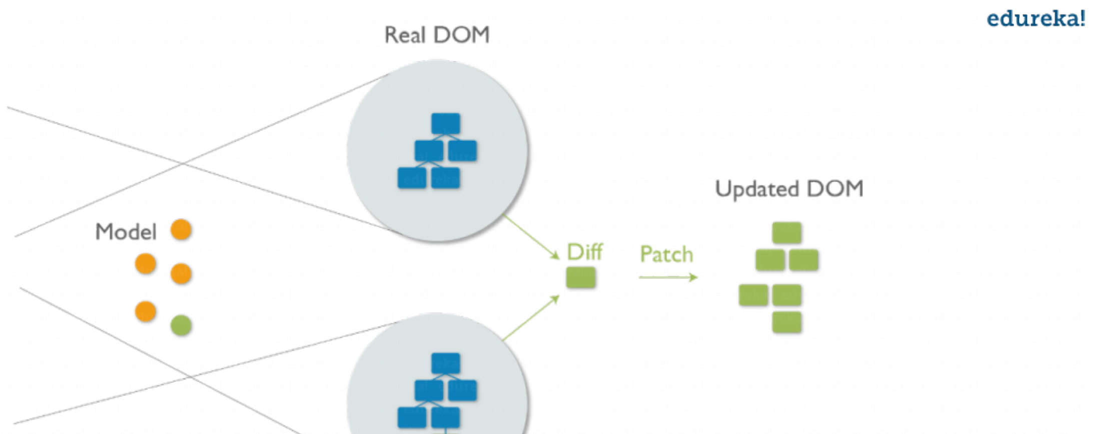
Figure*: ReactJS Tutorial – React Features*

Learning Curve

React has a shallow learning curve and it is suitable for beginners. ES6 syntax is easier to manage especially for smaller to-do apps. In React, you code in the 'JavaScript' way, giving you the freedom to choose your tool depending upon your need. Angular expects you to learn one additional tool 'typescript' which can be viewed as the 'Angular' way of doing things. In 'Angular' you need to learn the entire framework even if you're just building a simple UI application.

Moving ahead in this ReactJS tutorial, I will be discussing about React's Virtual DOM.

Concept: The Simplicity Of Virtual DOM



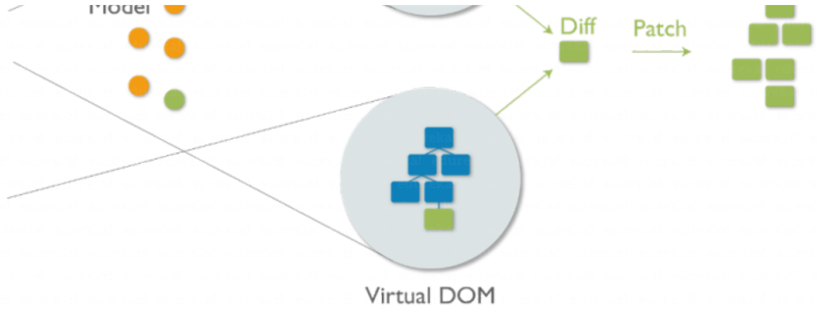


Figure : *ReactJS Tutorial – React Virtual DOM*

In contrary to the actual DOM, react makes use of the Virtual DOM. Virtual DOM utilizes a differential algorithm for making calculations. This relieves the real DOM which can then process other tasks. Let me illustrate this with an example.

Now consider there are 10,000 nodes out of which we only need to work on 2 nodes. Now most of the processing is wasted in traversing those 10,000 nodes while we only operate on 2 nodes. The calculations are done by the Virtual DOM to find those 2 nodes and the real DOM quickly retrieves them.

Performance

When it comes to performance, React sits right at the top. React is known for its superior rendering speed. Thus the name "React", an instant reaction to change with minimum delay. DOM manipulation is the heart of a responsive website, unfortunately it is slow in most JavaScript frameworks. However, Virtual DOM is implemented in React, hence it is the underlying principle behind React's superior performance.

Size

As we already know, React is not a framework, thus features may be added according to the user's needs. This is the principle behind the light-weight applications built on React – *pick only what is needed*. Webpack offers several plugins which further minimize (minify) the size during production, The React + Redux bundle minified constitutes around 200 kb whereas its rival Angular is almost four times bigger (Angular + RxJS bundle).

Debugging

There will be a point when a developer goes through a roadblock. It could be as simple as a 'missing bracket' or as tricky as a 'segmentation fault'. In any case, the earlier the exception is caught the lesser is the cost overhead. React uses compile time debugging and detects errors at an early stage. This ensures that errors don't silently turn up at run-time. Facebook's unidirectional data flow allows clean and smooth debugging, fewer stack traces, lesser clutter and an organized Flux architecture for bigger applications.

How Does It Work?

While React is easier to learn for beginners with no prior JavaScript experience, the nitty gritty's of transpiling JSX code can often be overwhelming. This sets the tone for tools such as Babel and Webpack. Webpack and Babel bundle together all the JavaScript files into a single file. Just like how we use to include a link to the CSS and JS files in our HTML code, Webpack performs a similar function eliminating the need for explicitly linking files.

I'm sure all of you use Facebook. Now, imagine Facebook being split into components, each functionality is assigned to a specific component and each component produces some HTML which is rendered as output by the DOM.

Facebook Components

- Search Bar

I'm sure all of you use Facebook. Now, imagine Facebook being split into components, each functionality is assigned to a specific component and each component produces some HTML which is rendered as output by the DOM.

Facebook Components

- Search Bar
- Add Post
- Notifications Bar
- Feed Updates
- Profile Info
- Chat Window

To make things clear, refer to the image below.

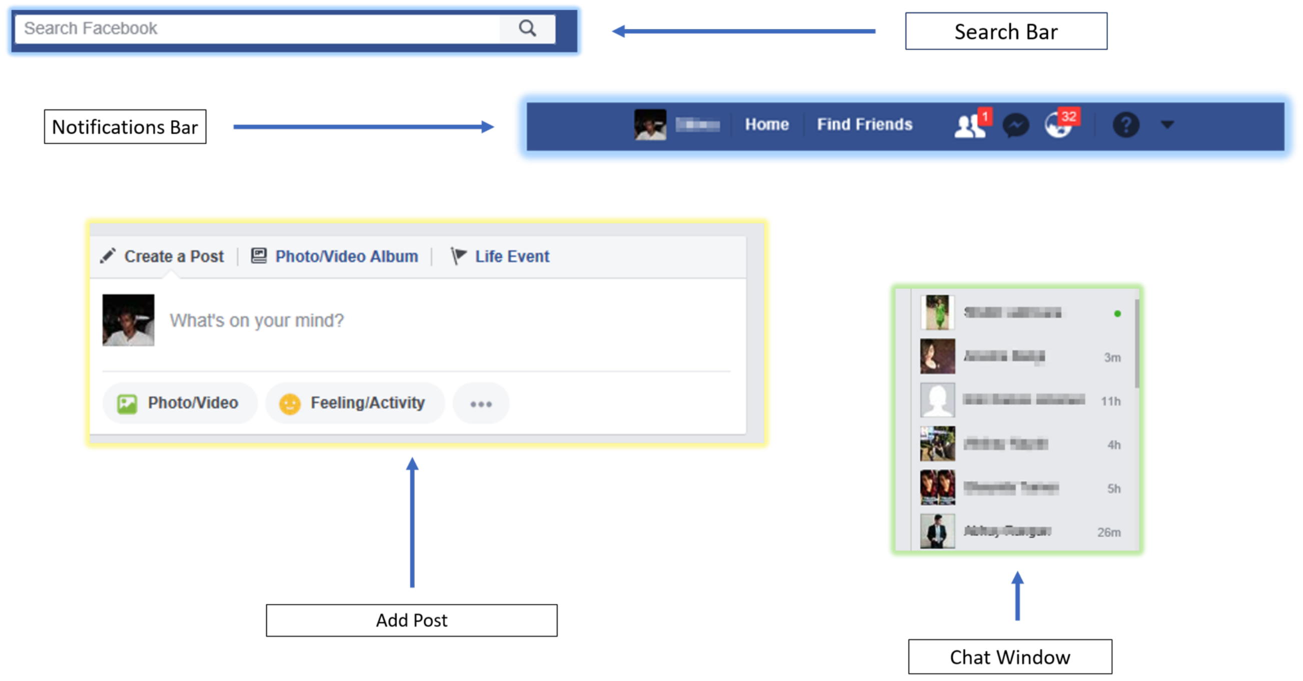


Figure: *ReactJS Tutorial – Facebook Components*

Building Blocks:-

- Components
- Props
- State
- State Lifecycle
- Event handling
- Keys

Moving on to the core aspect of our ReactJS tutorial, let us discuss the building blocks of React.

Components

The entire application can be modeled as a set of independent components. Different components are used to serve different purposes. This enables us to keep logic and views separate. React renders multiple components simultaneously. Components can be either stateful or stateless.

Before we start creating components, we need to include a few 'import' statements.

In the first line, we have to instruct JavaScript to import the 'react' library from the installed 'npm' module. This takes care of all the dependencies needed by React.

```
import React from 'react';
```

The HTML generated by the component needs to be displayed on to the DOM, we achieve this by

Before we start creating components, we need to include a few 'import' statements.

In the first line, we have to instruct JavaScript to import the 'react' library from the installed 'npm' module. This takes care of all the dependencies needed by React.

```
import React from 'react';
```

The HTML generated by the component needs to be displayed on to the DOM, we achieve this by specifying a render function which tells React where exactly it needs to be rendered (displayed) on the screen. For this, we make a reference to an existing DOM node by passing a container element.

In React, the DOM is part of the 'react-dom' library. So in the next line, we have to instruct JavaScript to import 'react-dom' library from the installed npm module.

```
import ReactDOM from 'react-dom';
```

In our example, we create a component named 'MyComponent' which displays a welcome message. We pass the component instance '<MyComponent\>' to React along with its container '<div >' tag.

```
const MyComponent =()=> {

      {     return
<h2>Way to go you just created a component!!</h2>

;


      }

}
ReactDOM.render(<MyComponent/>, document.getElementById('root'));
```

LEARN REACT FROM EXPERTS

Props

"All the user needs to do is, change the parent component's state, while the changes are passed down to the child component through props."

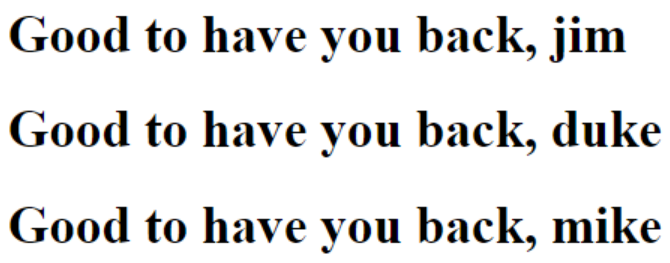
Props is a shorthand for properties (You guessed it right!). React uses 'props' to pass attributes from 'parent' component to 'child' component.

Props are the argument passed to the function or component which is ultimately processed by React. Let me illustrate this with an example.

```
function Message(props) {
    return


<h1>Good to have you back, {props.username}</h1>




;
}
function App() {
    return (



<div>
            <Message username="jim" />
            <Message username="duke" />
            <Message username="mike" />
        </div>




    );
}


    ReactDOM.render(
            <Message username="mike" />
        </div>



    );
}


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

Here the ‘App’ component has passed three ‘Message’ component instances with the prop ‘username’. All the three usernames are passed as an argument to the Message component.

The output screen is as shown below:



```
Good to have you back, jim

Good to have you back, duke

Good to have you back, mike
```

Figure: *ReactJS Tutorial – Props Output*

State

“And I believe state adds the greatest value to React.”

State allows us to create components that are dynamic and interactive. State is private, it must not be manipulated from the outside. Also, it is important to know when to use ‘state’, it is generally used with data that is bound to change. For example, when we click a toggle button it changes from ‘inactive’ to ‘active’ state. State is used only when needed, make sure it is used in render() otherwise don’t include it in the state. We do not use ‘state’ with static components. The state can only be set inside the constructor. Let’s include some code snippets to explain the same.

```
class Toggle extends React.Component {
constructor(value)
{
super(value);
this.state = {isToggleOn: true};
this.handleClick = this.handleClick.bind(this);
}
```

Binding is needed explicitly, as by default the event is not bound to the component.

Event Handling And Manipulation Of State

Whenever an event such as a button click or a mouse hover occurs, we need to handle these events and perform the appropriate actions. This is done using event **handlers**.

While State is set only once inside the constructor it can however, be manipulated through “setState” command. Whenever we call “handleclick” function based on the previous state, “isToggleOn” function is switched between “active” and “inactive” state.

```
handleClick()
{
this.setState(prevState =>({
isToggleOn: !prevState.isToggleOn
}));
}
```

The OnClick attribute specifies the function to be executed when the target element is clicked. In our example, whenever “onclick” is heard, we are telling React to transfer control to handleClick() which switches between the two states.

```
this.setState(prevState =>({
isToggleOn: !prevState.isToggleOn
}));
}
```

The OnClick attribute specifies the function to be executed when the target element is clicked. In our example, whenever “onclick” is heard, we are telling React to transfer control to handleClick() which switches between the two states.

```
render()
{
return(
<button onClick={this.handleClick}>
{this.state.isToggleOn ? 'ON': 'OFF'}

);
}
}// end class
```

State Lifecycle

We need to initialize resources to components according to their requirements. This is called “mounting” in React. It is critical to clear these resources taken by the components when they are destroyed. This is because performance can be managed and unused resources can be destroyed. This is called “unmounting” in React. It is not essential to use state lifecycle methods, but use them if you wish to take control of the complete resource allocations and retrieval process. State lifecycle methods component DidMount() and componentWillUnmount() are used to allocate and release resources respectively.

```
Class Time extends React.component
{
constructor(value) {
super(value);
this.state = {date: new Date()};
}
```

We create an object called Timer ID and set an interval of 2 seconds. Now, this is the time interval based on which the page is refreshed.

```
componentDidMount() {
this.timerID = setInterval( () => this.tick(),2000);
}
```

Here the interval is the timeframe after which the resources are cleared and the component should be destroyed. Performing such manipulations on the dataset using ‘state’ can be viewed as an optimal approach.

```
componentWillUnmount() {clear interval(this.timerID);}
```

A timer is set to call tick() method once every two seconds. An object with current Date is passed to set state. Each time React calls the render() method, **this.state.date** value is different. React then displays the updated time on the screen.

```
tick(){this.setState({date:new Date()});}
render()
{
return (



<div>



<h2>The Time is {this.state.date.toLocaleTimeString()}.</h2>



</div>




    );
  }
ReactDOM.render( <Time />, document.getElementById('root') );
}// end class
```

Keys

```



         );
     }
ReactDOM.render( <Time />, document.getElementById('root') );
}// end class
```

Keys

Keys in React provide identity to components. Keys are the means by which React identifies components uniquely. While working with individual components we don't need keys as react takes care of key assignment according to their rendering order. However, we need a strategy to differentiate between thousands of elements in a list. We assign them 'keys'. If we need to access the last component in a list using keys, it saves us from traversing the entire list sequentially. Keys serve to keep track of which items have been manipulated. Keys should be given to the elements inside the array to give the elements a stable identity.

In our example below, we create an array 'Data' with four items, we assign each item the index 'i' as the key. We achieve this by defining the key as a property('Prop') and use the JavaScript 'map' function to pass the key on each element of the array and return the result to the 'content' component.

```
class App extends React.Component {
constructor() {
super();

this.state = {
data:
[
{
item: 'Java',
id: '1'
},

{
item: 'React',
id: '2'
},

{
item: 'Python',
id: '3'
},
{
item: 'C#',
id: '4'
}
]
}
render() {
        return (



<div>



<div>
                    {this.state.data.map((dynamicComponent, i) => <Content key = {i}
                </div>




            </div>




        );
    }
}

class Content extends React.Component {
    render() {
        return (



<div>



<div>{this.props.componentData.component}</div>
```

```
    render() {
        return (



<div>



<div>{this.props.componentData.component}</div>






<div>{this.props.componentData.id}</div>



            </div>



        );

    }

}

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

React Installation

There are several ways to install React. In short, we can either configure the dependencies manually or use the open source starter packs available on GitHub. The 'create-react-app' (CRA) tool maintained by Facebook itself is one such example. This is suitable for beginners who can focus on code, without manually having to deal with transpiling tools like webpack and Babel. In this ReactJS tutorial I will be showing you how to install React using CRA tool.

Npm: Node Package Manager manages the different dependencies needed to run ReactJs applications. Npm is bundled together with node.js.

Step 1: Download NodeJS

First go to the node.js website and download the .exe file according to your system configuration and install it.

Link: https://nodejs.org/en/download/

Step 2: Download the 'create-react-app' Tool from GitHub

Link: https://github.com/facebookincubator/create-react-app

Step 3: Open cmd prompt and navigate to the project directory.

Now, enter the following commands

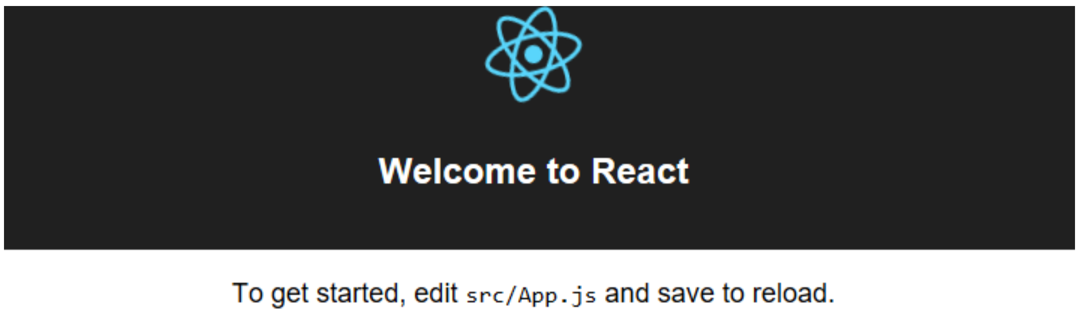
```
->  npm install -g create-react-app
->  cd my-app
->  create-react-app my-app
```

Step 4:-> npm start

Once we type "npm start" the application will start execution on port 3000. Open http://localhost:3000/ , you will be greeted by this page.

Step 4:-> npm start

Once we type "npm start" the application will start execution on port 3000. Open http://localhost:3000/ , you will be greeted by this page.



To get started, edit `src/App.js` and save to reload.

Figure: *ReactJS Tutorial – Welcome Page*

This is how the file structure should look once you have successfully installed React.

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

When you are creating new apps, all you need to do is update the file 'App.js' and the changes will be reflected automatically, other files can be added or removed. Make sure you put all CSS and JS files inside the '/src' directory.

This brings us to the end of this ReactJS tutorial blog. Hope each and every aspect I discussed above is clear to you to all. To learn more check out our courses on React.

MASTER REACT WITH EDUREKA

*If you found this blog on "**ReactJS tutorial**" relevant, check out the ReactJS with Redux Certification Training by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. This Edureka course helps learners gain expertise in both fundamental and advanced topics in React enabling you to develop full-fledged, dynamic web applications on the go.*

Got a question for us? Please mention it in the comments section and we will get back to you.

learning company with a network of more than 250,000 satisfied learners spread across the globe. This Edureka course helps learners gain expertise in both fundamental and advanced topics in React enabling you to develop full-fledged, dynamic web applications on the go.

Got a question for us? Please mention it in the comments section and we will get back to you.

About Shivaprakash (5 Posts)

Shivaprakash is a Research analyst at Edureka. He has expertise on front end web technologies like jQuery and React JS. He is also passionate about data science and exploring new technologies.