Introduction
What is ReactJS?

React Introduction

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library responsible only for the view layer of the application. It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

Today, most of the websites are built using MVC (model view controller) architecture. In MVC architecture, React is the 'V' which stands for view, whereas the architecture is provided by the Redux or Flux.

A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

To create React app, we write React components that correspond to various elements. We organize these components inside higher level components which define the application structure. For example, we take a form that consists of many elements like input fields, labels, or buttons. We can write each element of the form as React components, and then we combine it into a higher-level component, i.e., the form component itself. The form components would specify the structure of the form along with elements inside of it.

Why learn ReactJS?

Today, many JavaScript frameworks are available in the market (like angular, node), but still, React came into the market and gained popularity amongst them. The previous frameworks follow the traditional data flow structure, which uses the DOM (Document Object Model). DOM is an object which is created by the browser each time a web page is loaded. It dynamically adds or removes the data at the back end and when any modifications were done, then each time a new DOM is created for the same page. This repeated creation of DOM makes unnecessary memory wastage and reduces the performance of the application.

Therefore, a new technology ReactJS framework invented which remove this drawback. ReactJS allows you to divide your entire application into various components. ReactJS still used the same traditional data flow, but it is not directly operating on the browser's Document Object Model (DOM) immediately; instead, it operates on a virtual DOM. It means rather than manipulating the document in a browser after changes to our data, it resolves changes on a DOM built and run entirely in memory. After the virtual DOM has been updated, React determines what changes made to the actual browser's DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM; instead, we are writing virtual components that react will turn into the DOM.

React Version

A complete release history for React is given below. You can also see the full documentation for recent releases on GitHub

SN	Version	Release Date	Significant Changes
1.	0.3.0	29/05/2013	Initial Public Release
2.	0.4.0	20/07/2013	Support for comment nodes {/* */} , Improved server-side rendering APIs, Removed React.autoBind, Support for the key prop, Improvements to forms, Fixed bugs.
3.	0.5.0	20/10/2013	Improve Memory usage, Support for Selection and Composition events, Support for getInitialState and getDefaultProps in mixins, Added React.version and React.isValidClass, Improved compatibility for Windows.
4.	0.8.0	20/12/2013	Added support for rows & cols, defer & async, loop for <audio> & <video>, autoCorrect attributes. Added onContextMenu events, Upgraded jstransform and esprima-fb tools, Upgraded browserify.
5.	0.9.0	20/02/2014	Added support for crossOrigin, download and hrefLang, mediaGroup and muted, sandbox, seamless, and srcDoc, scope attributes, Added any, arrayOf, component, oneOfType, renderable, shape to React.PropTypes, Added support for onMouseOver and onMouseOut event, Added support for onLoad and onError on <img> elements.
6.	0.10.0	21-03-2014	Added support for srcSet and textAnchor attributes, add update function for immutable data, Ensure all void elements don't insert a closing tag.
7.	0.11.0	17/07/2014	Improved SVG support, Normalized e.view event, Update $apply command, Added support for namespaces, Added new transformWithDetails API, includes pre-built packages under dist/, MyComponent() now returns a descriptor, not an instance.
8.	0.12.0	21/11/2014	Added new features Spread operator ({...}) introduced to deprecate this.transferPropsTo, Added support for acceptCharset, classID, manifest HTML attributes, React.addons.batchedUpdates added to API, @jsx React.DOM no longer required, Fixed issues with CSS Transitions.
9.	0.13.0	10/03/2015	Deprecated patterns that warned in 0.12 no longer work, ref resolution order has changed, Removed properties this._pendingState and this._rootNodeID, Support ES6 classes, Added API React.findDOMNode(component), Support for iterators and immutable-js sequences, Added new features React.addons.createFragment, deprecated React.addons.classSet.
10.	0.14.1	29/10/2015	Added support for srcLang, default, kind attributes, and color attribute, Ensured legacy .props access on DOM nodes, Fixed scryRenderedDOMComponentsWithClass, Added react-dom.js.

11.	15.0.0	07/04/2016	Initial render now uses document.createElement instead of generating HTML, No more extra s, Improved SVG support, ReactPerf.getLastMeasurements() is opaque, New deprecations introduced with a warning, Fixed multiple small memory leaks, React DOM now supports the cite and profile HTML attributes and cssFloat, gridRow and gridColumn CSS properties.
12.	15.1.0	20/05/2016	Fix a batching bug, Ensure use of the latest object-assign, Fix regression, Remove use of merge utility, Renamed some modules.
13.	15.2.0	01/07/2016	Include component stack information, Stop validating props at mount time, Add React.PropTyjpes.symbol, Add onLoad handling to <link> and onError handling to <source> element, Add isRunning() API, Fix performance regression.
14.	15.3.0	30/07/2016	Add React.PureComponent, Fix issue with nested server rendering, Add xmlns, xmlnsXlink to support SVG attributes and refererPolicy to HTML attributes, updates React Perf Add-on, Fixed issue with ref.
15.	15.3.1	19/08/2016	Improve performance of development builds, Cleanup internal hooks, Upgrade fbjs, Improve startup time of React, Fix memory leak in server rendering, fix React Test Renderer, Change trackedTouchCount invariant into a console.error.
16.	15.4.0	16/11/2016	React package and browser build no longer includes React DOM, Improved development performance, Fixed occasional test failures, update batchedUpdates API, React Perf, and ReactTestRenderer.create().
17.	15.4.1	23/11/2016	Restructure variable assignment, Fixed event handling, Fixed compatibility of browser build with AMD environments.
18.	15.4.2	06/01/2017	Fixed build issues, Added missing package dependencies, Improved error messages.
19.	15.5.0	07/04/2017	Added react-dom/test-utils, removed peerDependencies, Fixed issue with Closure Compiler, Added a deprecation warning for React.createClass and React.PropTypes, Fixed Chrome bug.
20.	15.5.4	11/04/2017	Fix compatibility with Enzyme by exposing batchedUpdates on shallow renderer, Update version of prop-types, Fix react-addons-create-fragment package to include loose-envify transform.
21.	15.6.0	13/06/2017	Add support for CSS variables in style attribute and Grid style properties, Fix AMD support for addons depending on react, remove unnecessary dependency, Add a deprecation warning for React.createClass and React.DOM factory helpers.
22.	16.0.0	26/09/2017	Improvd error handling with introduction of "error boundaries", React DOM allows passing non-standard attributes, Minor changes to setState behavior, remove react-with-addons.js build, Add React.createClass as create-react-class, React.PropTypes as prop-types, React.DOM as react-dom-factories, changes to the behavior of scheduling and lifecycle methods.

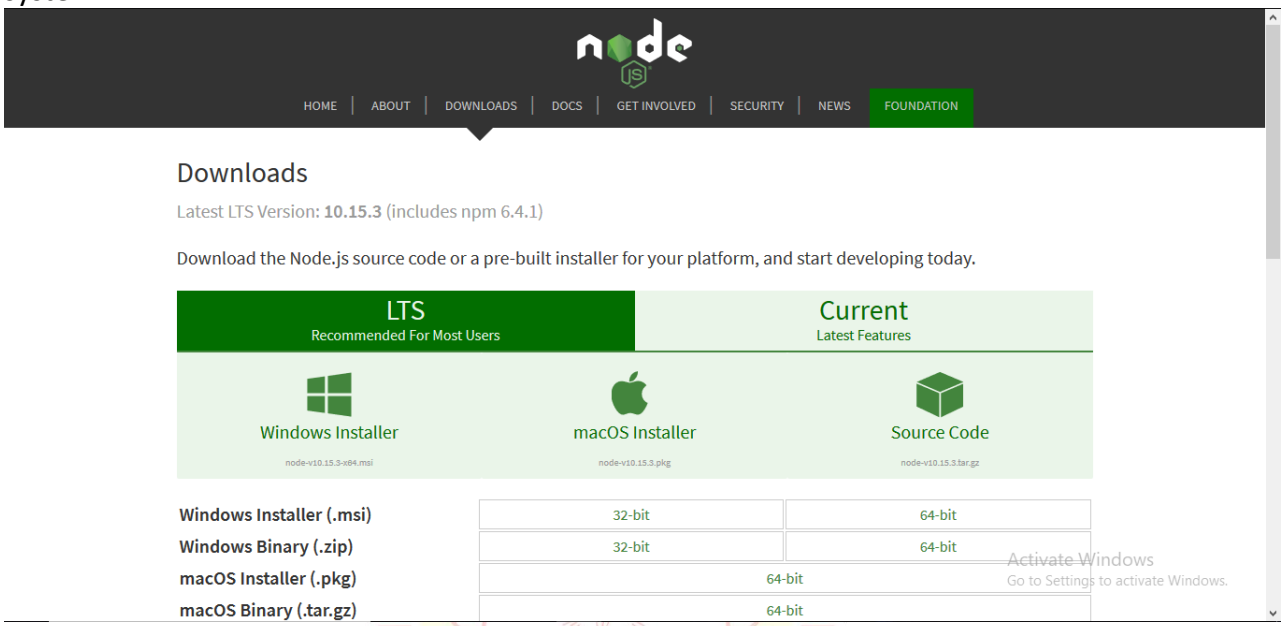
23.	16.1.0	9/11/2017	Discontinuing Bower Releases, Fix an accidental extra global variable in the UMD builds, Fix onMouseEnter and onMouseLeave firing, Fix <textarea> placeholder, Remove unused code, Add a missing package.json dependency, Add support for React DevTools.
24.	16.3.0	29/03/2018	Add a new officially supported context API, Add new packagePrevent an infinite loop when attempting to render portals with SSR, Fix an issue with this.state, Fix an IE/Edge issue.
25.	16.3.1	03/04/2018	Prefix private API, Fix performance regression and error handling bugs in development mode, Add peer dependency, Fix a false positive warning in IE11 when using Fragment.
26.	16.3.2	16/04/2018	Fix an IE crash, Fix labels in User Timing measurements, Add a UMD build, Improve performance of unstable_observedBits API with nesting.
27.	16.4.0	24/05/2018	Add support for Pointer Events specification, Add the ability to specify propTypes, Fix reading context, Fix the getDerivedStateFromProps() support, Fix a testInstance.parent crash, Add React.unstable_Profiler component for measuring performance, Change internal event names.
28.	16.5.0	05/09/2018	Add support for React DevTools Profiler, Handle errors in more edge cases gracefully, Add react-dom/profiling, Add onAuxClick event for browsers, Add movementX and movementY fields to mouse events, Add tangentialPressure and twist fields to pointer event.
29.	16.6.0	23/10/2018	Add support for contextType, Support priority levels, continuations, and wrapped callbacks, Improve the fallback mechanism, Fix gray overlay on iOS Safari, Add React.lazy() for code splitting components.
30.	16.7.0	20/12/2018	Fix performance of React.lazy for lazily-loaded components, Clear fields on unmount to avoid memory leaks, Fix bug with SSR, Fix a performance regression.
31.	16.8.0	06/02/2019	Add Hooks, Add ReactTestRenderer.act() and ReactTestUtils.act() for batching updates, Support synchronous thenables passed to React.lazy(), Improve useReducer Hook lazy initialization API.
32.	16.8.6	27/03/2019	Fix an incorrect bailout in useReducer(), Fix iframe warnings in Safari DevTools, Warn if contextType is set to Context.Consumer instead of Context, Warn if contextType is set to invalid values.

Installation or Setup

Node JS Installation

Step-1: Downloading the Node.js '.msi' installer.

The first step to install Node.js on windows is to download the installer. Visit the official Node.js website i.e) https://nodejs.org/en/download/ and download the .msi file according to your system environment (32-bit & 64-bit). An MSI installer will be downloaded on your system.



Step-2: Running the Node.js installer.

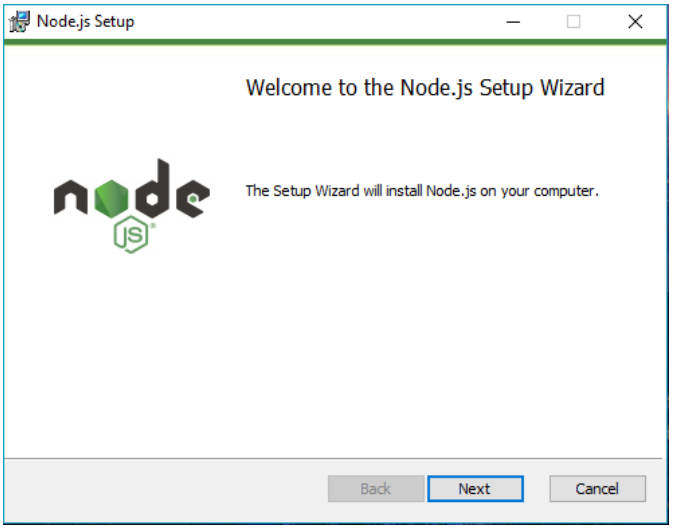
Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:-

- Double click on the .msi installer.

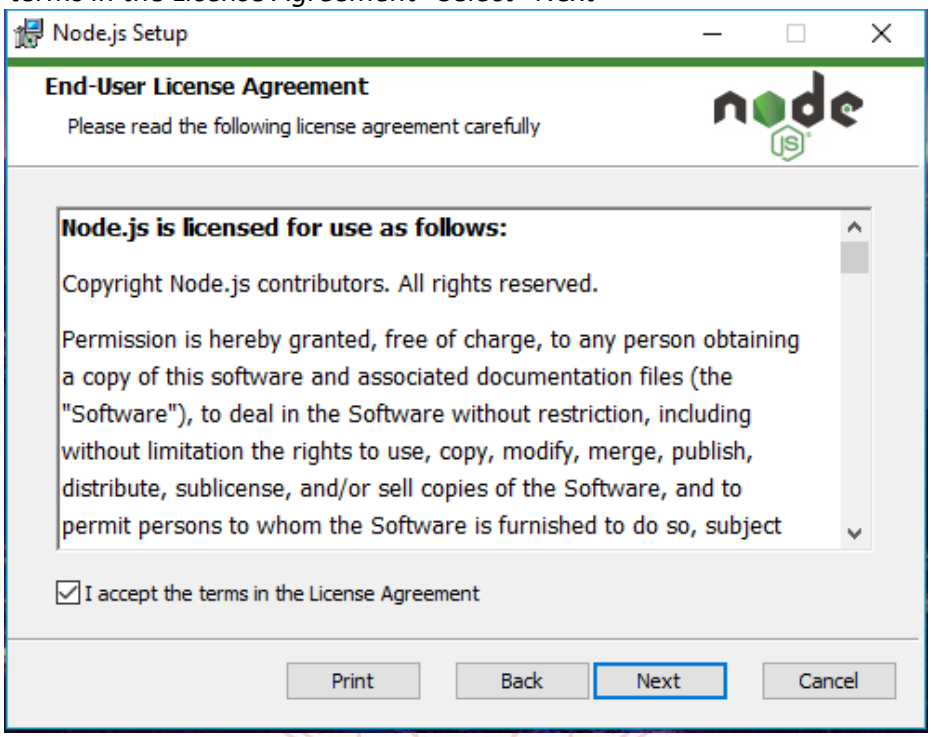
The Node.js Setup wizard will open.

- Welcome To Node.js Setup Wizard.

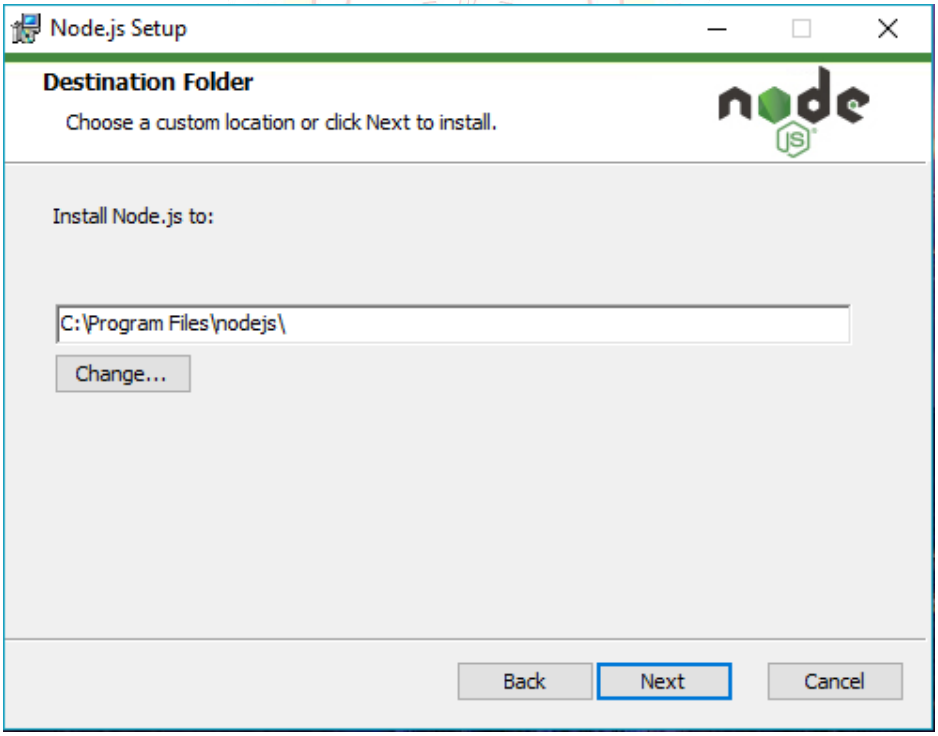
Select "Next"



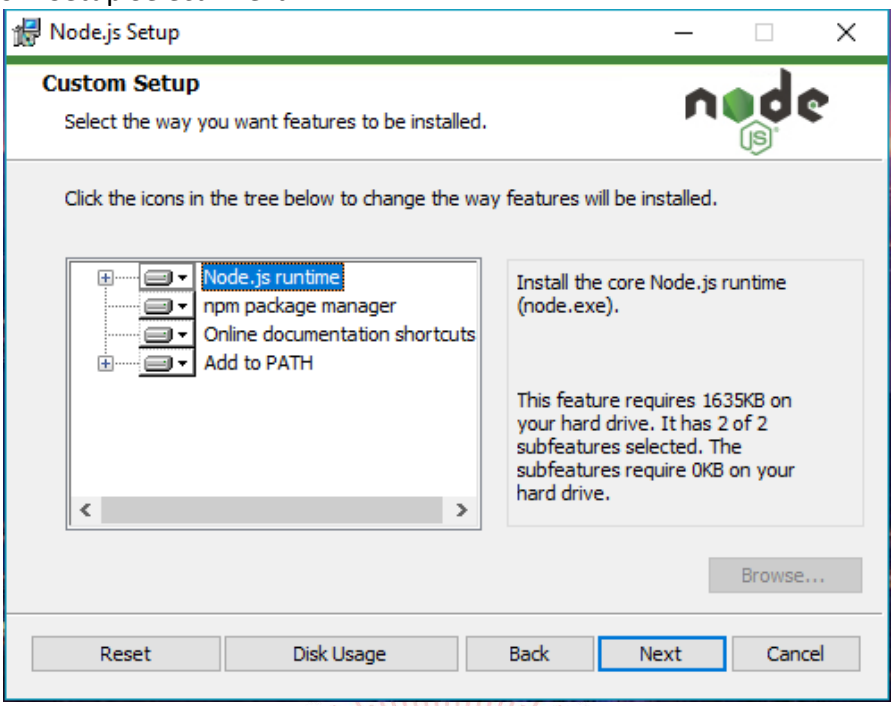
- After clicking “Next”, End-User License Agreement (EULA) will open. *Check “I accept the terms in the License Agreement” Select “Next”*



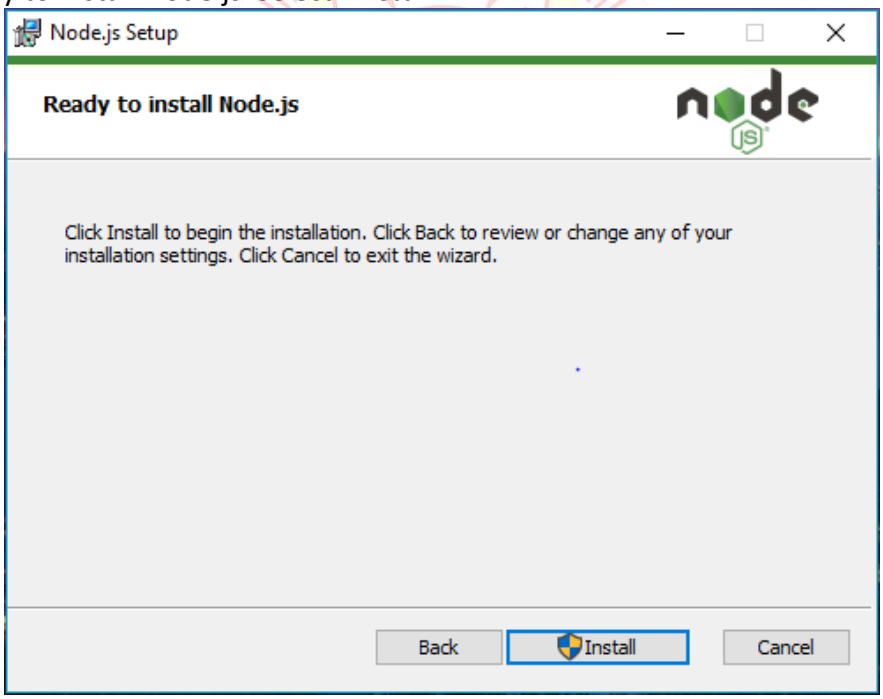
- Destination Folder *Set the Destination Folder where you want to install Node.js & Select “Next”*



- Custom Setup *Select "Next"*



- Ready to Install Node.js. *Select "Install"*

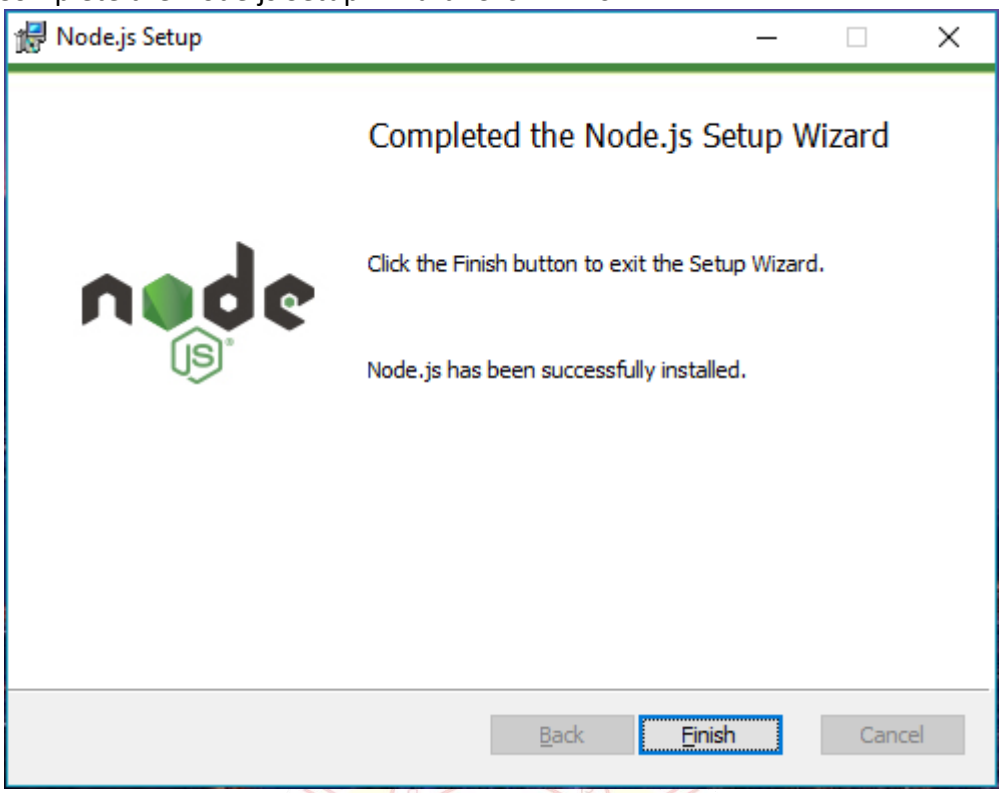


NOTE :
A prompt saying – "This step requires administrative privileges" will appear.
Authenticate the prompt as an "Administrator"

- Installing Node.js.

Do not close or cancel the installer until the install is complete

- Complete the Node.js Setup Wizard. *Click "Finish"*



To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it: -

C:\Users\HardikCHavda> node -v

Next, we will learn how to set up an environment for the successful development of ReactJS application.

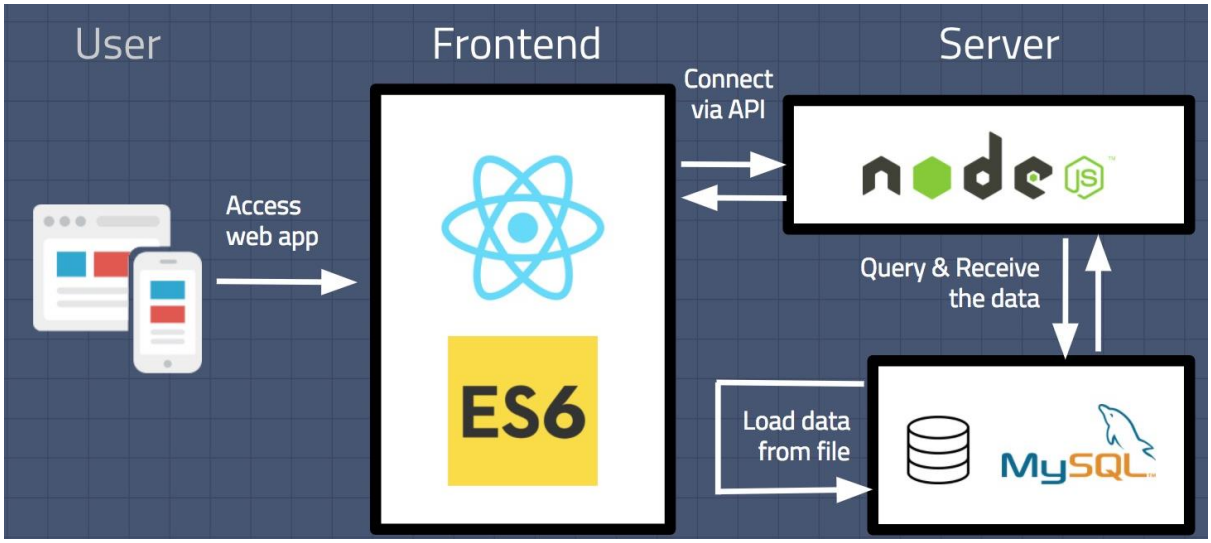
Ways to install ReactJS

There are two ways to set up an environment for successful ReactJS application. They are given below.

Using the npm command

Using the create-react-app command

1. Using the npm command
2. Using the npx command



NPM (Node Package Manager) is a package manager, but it's not very good at executing (running) packages.

NPX (Node Package Execute) is a package-runner CLI tool that is built-in to NPM (since NPM version 5.2).

You'll notice that the official documentation for ReactJS recommends you to use this NPX command to install and run create-react-app:

```
npx create-react-app project-name
```

The reason is that NPX makes sure that you run the React application with the latest versions of the packages — NPM doesn't.

This is practical because **create-react-app** is updated frequently — and it's generally recommended to use the latest package versions.

If you install create-react-app globally with npm install, you'll have to manually update your project every time packages/dependencies get updated. NPX does it for you automatically.

NodeJS and NPM are the platforms need to develop any ReactJS application. You can install NodeJS from **https://nodejs.org/en/**

You can install React using npm package manager by using the below command. There is no need to worry about the complexity of React installation. The create-react-app npm package will take care of it.

```
D:\HardikChavda>npm install -g create-react-app
```

After the installation of React, you can create a new react project using create-react-app command. Here, I choose myapp name for my project.

```
D:\HardikChavda>cd myapp
D:\HardikChavda\myapp>
```

NOTE: You can combine the above two steps in a single command using npx. The npx is a package runner tool that comes with npm 5.2 and above version.

```
D:\HardikChavda>npx create-react-app myapp
```

The above command will install the react and create a new project with the name myapp. This app contains the following sub-folders and files by default which can be shown in the below image.

Now, to get started, open the src folder and make changes in your desired file. By default, the src folder contain the following files shown in below image.

For example, I will open App.js and make changes in its code which are shown below.
App.js

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Welcome To Geetanjali College.
          <p>To get started, edit src/App.js and save to reload.</p>
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

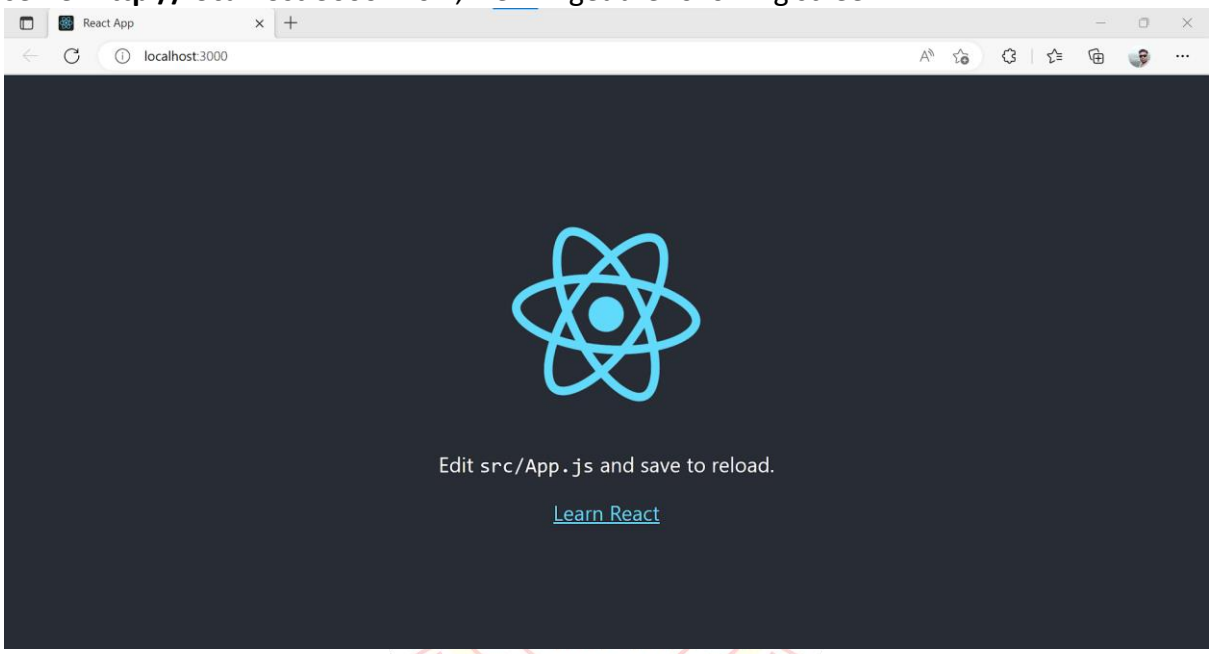
Running the Server

After completing the installation process, you can start the server by running the following command.

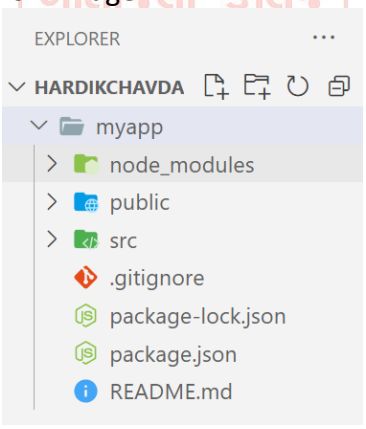
```
D:\HardikChavda\myapp>npm start
```

It will show the port number which we need to open in the browser.

NPM is a package manager which starts the server and access the application at default server **http://localhost:3000**. Now, we will get the following screen.



Next, open the project on Code editor. Here, I am using Visual Studio Code. Our project's default structure looks like as below image.



In React application, there are several files and folders in the root directory. Some of them are as follows:

node_modules: It contains the React library and any other third party libraries needed.

public: It holds the public assets of the application. It contains the index.html where React will mount the application by default on the element.

src: It contains the **App.css, App.js, App.test.js, index.css, index.js**, etc. files. Here, the **Index.js** file always responsible for displaying the output screen in React.

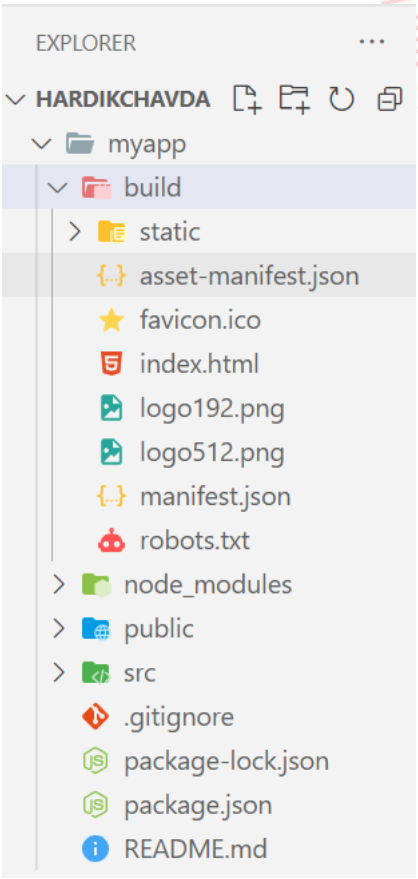
package-lock.json: It is generated automatically for any operations where npm package modifies either the node_modules tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.

package.json: It holds various metadata required for the project. It gives information to npm, which allows to identify the project as well as handle the project's dependencies.

README.md: It provides the documentation to read about React topics.

Next, if we want to make the project for the production mode, type the following command. This command will generate the production build, which is best optimized.

```
D:\HardikChavda\myapp>npm run build
```



For the project to build, these files must exist with exact filenames:

public/index.html is the page template;
src/index.js is the JavaScript entry point.
You can delete or rename the other files.

You may create subdirectories inside src. For faster rebuilds, only files inside src are processed by **webpack**. You need to put any JS and CSS files inside src, otherwise webpack won't see them.

You can, however, create more top-level directories. They will not be included in the production build so you can use them for things like documentation.

Components:

Earlier, the developers write more than thousands of lines of code for developing a single page application. These applications follow the traditional DOM structure, and making changes in them was a very challenging task. If any mistake found, it manually searches the entire application and update accordingly. The component-based approach was introduced to overcome an issue. In this approach, the entire application is divided into a small logical group of code, which is known as components.

Every React component have their own structure, methods as well as APIs. They can be reusable as per your need. For better understanding, consider the entire UI as a tree. Here, the root is the starting component, and each of the other pieces becomes branches, which are further divided into sub-branches.

Creating components

A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.

Basic components,

Higher order components are used when you want to share logic across several components regardless of how different they render.

```
import logo from './logo.svg';
import './App.css';

function App() { //Component
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

Given the following file: index.JS

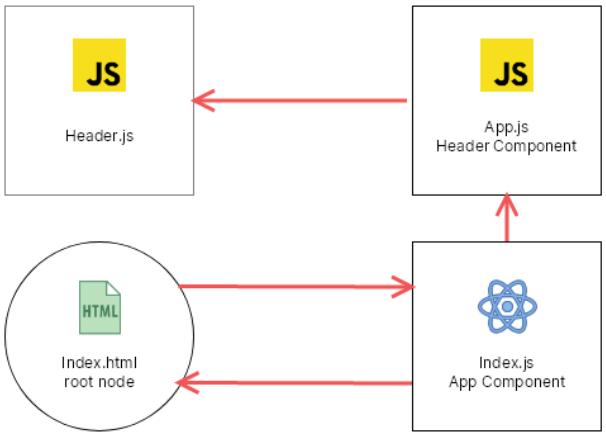
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';  //Getting Component

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
   <App /> //Using Component
);
```

Nesting components,

A lot of the power of ReactJS is its ability to allow nesting of components. Take the following two components.

Header.js is a component.

```
function Header() {
   return (
      <Header>This is Header</Header>
   )
}
export default Header
```



Header is imported and added in App.js

```
import Header from "./Header";
function App() {
  return (
    <Header />
  );
}
export default App;
```

App.js is imported into index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
   <App />
);
```

Functional Component:

In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered. A valid functional component can be shown in the below example.

```
function WelcomeMessage() {
   return <h1>Welcome to the New Component</h1>;
}
```

The functional component is also known as a stateless component because they do not hold or manage state. It can be explained in the below example.

Class Component

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function. Valid class component is shown in the below example.

```
class MyComponent extends React.Component {
   render() {
      return <div>This is main component. </div>
   }
}
```

The class component is also known as a stateful component because they can hold or manage local state. It can be explained in the below example.

In this example, we are creating the list of unordered elements, where we will dynamically insert StudentName for every object from the data array. Here, we are using ES6 arrow syntax (=>) which looks much cleaner than the old JavaScript syntax. It helps us to create our elements with fewer lines of code. It is especially useful when we need to create a list with a lot of items.

```
import React, { Component } from 'react';
class App extends React.Component {
   constructor() {
      super();
      this.state = {
         data:
            [{ "name": "Abhishek" }, { "name": "Saharsh" }, { "name": "Ajay" }]
      }
   }
   render() {
      return (
         <div>
            <StudentName />
            <ul>
               {this.state.data.map((item) => <List data={item} />)}
```

```
            </ul>
          </div>
        );
    }
}
class StudentName extends React.Component {
    render() {
        return (
            <div>
                <h1>Student Name Detail</h1>
            </div>
        );
    }
}
class List extends React.Component {
    render() {
        return (
            <ul>
                <li>{this.props.data.name}</li>
            </ul>
        );
    }
}
export default App;
```

Introduction to JSX:

As we have already seen that, all of the React components have a render function. The render function specifies the HTML output of a React component. JSX (JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML. In other words, JSX is an HTML-like syntax used by React that extends **ECMAScript** so that HTML-like syntax can co-exist with JavaScript/React code. The syntax is used by preprocessors (i.e., transpilers like babel) to transform HTML-like syntax into standard JavaScript objects that a JavaScript engine will parse.

JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

Here, we will write JSX syntax in JSX file and see the corresponding JavaScript code which transforms by preprocessor(**babel**).

JSX File

```
<div>Hello Geetanjali</div>
```

Corresponding Output

```
React.createElement("div", null, "Hello Geetanjali");
```

The above line creates a react element and passing three arguments inside where the first is the name of the element which is div, second is the attributes passed in the div tag, and last is the content you pass which is the "Hello Geetanjali."

Why use JSX?

It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both. We will learn components in a further section.
It is type-safe, and most of the errors can be found at compilation time.
It makes easier to create templates.

Nested Elements in JSX

To use more than one element, you need to wrap it with one container element. Here, we use div as a container element which has three nested elements inside it.

```
import React, { Component } from 'react';
class App extends Component {
  render() {
    return (
      <div>
        <h1>Geetanjali</h1>
        <h2>Hardik Chavda</h2>
      </div>
    );
```

```
  }
}
export default App;
```

JSX Attributes

JSX use attributes with the HTML elements same as regular HTML. JSX uses camelcase naming convention for attributes rather than standard naming convention of HTML such as a class in HTML becomes className in JSX because the class is the reserved keyword in JavaScript. We can also use our own custom attributes in JSX. For custom attributes, we need to use data- prefix. In the below example, we have used a custom attribute data-demoAttribute as an attribute for the <p> tag.

In JSX, we can specify attribute values in two ways:

1. As String Literals: We can specify the values of attributes in double quotes:
var element = <h2 className = "firstAttribute">Hello Geetanjali</h2>;
Example

```
import React, { Component } from 'react';
class App extends Component {
  render() {
    return (
      <div>
        <h1 className="hello" >Geetanjali</h1>
        <p data-demoAttribute="demo">Hardik Chavda </p>
      </div>
    );
  }
}
export default App;
```

Output:
Geetanjali
Hardik Chavda

2. As Expressions: We can specify the values of attributes as expressions using curly braces {}:
var element = <h2 className = {varName}>Hello Geetanjali</h2>;
Example

```
import React, { Component } from 'react';
class App extends Component {
  render() {
    return (
      <div>
        <h1 className="hello" >{25 + 20}</h1>
      </div>
    );
  }
}
```

JSX Comments

JSX allows us to use comments that begin with /* and ends with */ and wrapping them in curly braces {} just like in the case of JSX expressions. Below example shows how to use comments in JSX.

Example

```
import React, { Component } from 'react';
class App extends Component {
   render() {
      return (
         <div>
            <h1 className="hello" >Hello Geetanjali</h1>
            {/* This is a comment in JSX */}
         </div>
      );
   }
}
export default App;
```

JSX Styling

React always recommends to use inline styles. To set inline styles, you need to use camelCase syntax. React automatically allows appending px after the number value on specific elements. The following example shows how to use styling in the element.

Example

```
import React, { Component } from 'react';
class App extends Component {
   render() {
      var myStyle = {
         fontSize: 80,
         fontFamily: 'Courier',
         color: '#003300'
      }
      return (
         <div>
            <h1 style={myStyle}>www.geetanjali.com</h1>
         </div>
      );
   }
}
export default App
```

ReactJS JSX

NOTE: JSX cannot allow to use if-else statements. Instead of it, you can use conditional (ternary) expressions. It can be seen in the following example.

Example

```
import React, { Component } from 'react';
class App extends Component {
   render() {
      var i = 5;
      return (
         <div>
            <h1>{i == 1 ? 'True!' : 'False!'}</h1>
         </div>
      );
   }
}
export default App;
```

Props: ReactJS Props,
Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

When you need immutable data in the component, you have to add props to reactDom.render() method in the main.js file of your ReactJS project and used it inside the component in which you need.

App.js

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
            <h1> Welcome to {this.props.name} </h1>
            <p> <h4> Geetanjali is one of the best College in India. </h4> </p>
         </div>
      );
   }
}
export default App;
```

Main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

ReactDOM.render(
         <App name="Geetanjali!!" />, document.getElementById('app')
);
```

Default Props

It is not necessary to always add props in the reactDom.render() element. You can also set default props directly on the component constructor.

App.js

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Default Props Example</h1>
            <h3>Welcome to {this.props.name}</h3>
            <p>It is one of the best College in India.</p>
         </div>
      );
   }
}
App.defaultProps = {
   name: "Geetanjali"
}
export default App;
```

Main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

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

React State,
Components, we got to know that React Components can be broadly classified into Functional and Class Components. It is also seen that Functional Components are faster and much simpler than Class Components. The primary difference between the two is the availability of the State.

What is State?
The state is an instance of React Component Class can be defined as an object of a set of observable properties that control the behavior of the component. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component. For example, let us think of the clock that we created in this article, we were calling the render() method every second explicitly, but React provides a better way to achieve the same result and that is by using State, storing the value of time as a member of the component's state. We will look into this more elaborately later in the article.

Difference of Props and State.
We have already learned about Props and we got to know that Props are also objects that hold information to control the behavior of that particular component, sounds familiar to State indeed but props and states are nowhere near be same. Let us differentiate the two.
Props are immutable i.e. once set the props cannot be changed, while State is an observable object that is to be used to hold data that may change over time and to control the behavior after each change.

States can be used in Class Components, Functional components with the use of React Hooks (useState and other methods) while Props don't have this limitation.

While Props are set by the parent component, State is generally updated by event handlers. It can be implemented using State where the probable values of the State can be either light or dark and upon selection, the IDE changes its color.

Now we have learned the basics of State and are able to differentiate it from Props. We have also seen a few places where we can use State now all that is left is to know about the basic conventions of using the React State before implementing one for ourselves.

Conventions of Using State in React:

State of a component should prevail throughout the lifetime, thus we must first have some initial state, to do so we should define the State in the constructor of the component's class. To define a state of any Class we can use the sample format below.

```
class MyClass extends React.Component
{
   constructor(props){
      super(props);
      this.state = { attribute: "value" };
   }
}
```

State should never be updated explicitly. React uses an observable object as the state that observes what changes are made to the state and helps the component behave accordingly. For example, if we update the state of any component like the following the webpage will not re-render itself because React State will not be able to detect the changes made.

```
this.state.attribute = "new-value";
```

Thus, React provides its own method setState(). setState() method takes a single parameter and expects an object which should contain the set of values to be updated. Once the update is done the method implicitly calls the render() method to repaint the page. Hence, the correct method of updating the value of a state will be similar to the code below.

```
this.setState({attribute: "new-value"});
```

The only time we are allowed to define the state explicitly is in the constructor to provide the initial state.

React is highly efficient and thus uses asynchronous state updates i.e. React may update multiple setState() updates in a single go. Thus using the value of the current state may not always generate the desired result. For example, let us take a case where we must keep a count (Likes of a Post). Many developers may miswrite the code as below.

```
this.setState({counter: this.state.count + this.props.diff});
```

Now due to asynchronous processing, this.state.count may produce an undesirable result. A more appropriate approach would be to use the following.

```
this.setState((prevState, props) => ({
        counter: prevState.count + props.diff
}));
```

In the above code we are using the ES6 thick arrow function format to take the previous state and props of the component as parameters and are updating the counter. The same can be written using the default functional way as follows.

```
this.setState(function(prevState, props){
        return {counter: prevState.count + props.diff};
});
```

State updates are independent. The state object of a component may contain multiple attributes and React allows to use setState() function to update only a subset of those attributes as well as using multiple setState() methods to update each attribute value independently. For example, let us take the following component state into account.

```
this.state = {
        darkTheme: False,
        searchTerm: ''};
```

The above definition has two attributes we can use a single setState() method to update both together, or we can use separate setState() methods to update the attributes independently. React internally merges setState() methods or updates only those attributes which are needed.

Destructing Props and State,
Destructuring is a simple property that is used to make code much clear and readable, mainly when we pass props in React.

What is Destructuring?

- Destructuring is a characteristic of JavaScript, It is used to take out sections of data from an array or objects, We can assign them to new own variables created by the developer.
- In destructuring, It does not change an array or any object, it makes a copy of the desired object or array element by assigning them in its own new variables, later we can use this new variable in React (class or functional) components.
- It makes the code more clear. When we access the props using this keyword, we have to use this/ this.props throughout the program, but by the use of restructuring, we can discard this/ this.props by assigning them in new variables.
- This is very difficult to monitor props in complex applications, so by assigning these props in new own variables we can make a code more readable.

Advantages of Destructuring:

- It makes developer's life easy, by assigning their own variables.
- Nested data is more complex, it takes time to access, but by the use of destructuring, we can access faster of nested data.
- It improves the sustainability, readability of code.
- It helps to cut the amount of code used in an application.
- It trims the number of steps taken to access data properties.
- It provides components with the exact data properties.
- It saves time from iterate over an array of objects multiple times.
- In ReactJS We use multiple times ternary operators inside the render function, without destructuring it looks complex and hard to access them, but by the use of destructuring, we can improve the readability of ternary operators.

How to use Destructuring?

We can use the Destructuring in the following method in ReactJS:

In this example, we are going to simply display some words using destructuring and without destructuring.

App.js:

```
import React from "react"
import Greet from './component/Greet'
```

```
class App extends React.component {
   render() {
      return (
         <div className="App">
            <Greet active="Hardik Chavda" activeStatus="CSE" />
         </div>
      );
   }
}
export default App;
```

Without Destructuring:

```
import React from 'react';

const Greet = props => {
   return (
      <div>
         <div className="XYZ">
            <h3> {props.active} </h3>
         </div>

         <div className="PQR">
            <h1>{props.activeStatus}</h1>
         </div>
      </div>
   )
}
export default Greet;
```

With Destructuring:

```
import React from 'react';

const Greet = () => {
   const {active, activeStatus } = props
   return (
      <div>
         <div >
            <h3> {active} </h3>
         </div>

         <div >
            <h1>{activeStatus}</h1>
         </div>
      </div>
   )
}
export default Greet;
```

setState,

All the React components can have a state associated with them. The state of a component can change either due to a response to an action performed by the user or an event triggered by the system. Whenever the state changes, React re-renders the component to the browser. Before updating the value of the state, we need to build an initial state setup. Once we are done with it, we use the setState() method to change the state object. It ensures that the component has been updated and calls for re-rendering of the component.

setState is asynchronous call means if synchronous call gets called it may not get updated at right time like to know current value of object after update using setState it may not get give current updated value on console. To get some behavior of synchronous need to pass function instead of object to setState.

Syntax: We can use setState() to change the state of the component directly as well as through an arrow function.

```
setState({ stateName: updatedStateValue })

// OR
setState((prevState) => ({
  stateName: prevState.stateName + 1
}))
```

Example

```
import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)

    // Set initial state
    this.state = {
      greeting:
        'Click the button to receive greetings'
    }

    // Binding this keyword
    this.updateState = this.updateState.bind(this)
  }

  updateState() {
    // Changing state
    this.setState({
      greeting:
        'HardikChavda welcomes you !!'
    })
  }
```

```
  render() {
    return (
      <div>
        <h2>Greetings Portal</h2>
        <p>{this.state.greeting}</p>

        {/* Set click handler */}
        <button onClick={this.updateState}>
          Click me!
        </button>
      </div>
    )
  }
}

export default App;
```