REACT JS

New Notes

By: Mr.Mahesh

ReactJS Tutorial Index

1. React Introduction
2. React Version
3. React Installation
4. create-react-app
5. React Features
6. Pros & Cons
7. ReactJS vs AngularJS
8. ReactJS vs ReactNative
9. React vs Vue
10. React JSX
11. React Components
12. React State
13. React Props
14. React Props Validation
15. React State vs Props
16. React Constructor
17. React Component API
18. Component Life Cycle
19. React Forms
20. Controlled vs Uncontrolled
21. React Events
22. Conditional Rendering
23. React Lists
24. React Keys
25. React Refs
26. React Fragments
27. React Router
28. React CSS
29. React Animation

30.React Bootstrap
31.React Map
32.React Table
33.Higher-Order Components
34.React Code Splitting
35.React Context
36.React Hooks
37.React Flux Concept
38.React Flux Vs MVC
39.React Redux
40.React Redux Example
41.React Portals
42.React Error Boundaries

Interview Questions

o React Interview Questions

2.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.

3. 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 <div>{/* */}</div>, 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 <span>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.PropTypes.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 referrerPolicy 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	Improved error handling with introduction of "error boundaries", React DOM allows passing non-standard attributes, Minor changes to setState behaviour, 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 behaviour 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 readingcontext, 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.

4. React Environment Setup

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

Pre-requisite for ReactJS

1. NodeJS and NPM
2. React and React DOM
3. Webpack
4. Babel

Ways to install ReactJS

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

1. Using the npm command
2. Using the create-react-app command

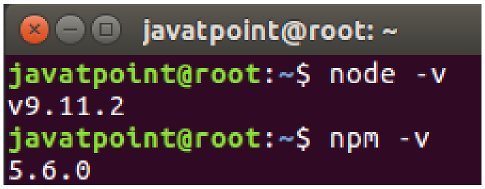
1. Using the npm command

Install NodeJS and NPM

NodeJS and NPM are the platforms need to develop any ReactJS application. You can install NodeJS and NPM package manager by the link given below.

https://www.javatpoint.com/install-nodejs-on-linux-ubuntu-centos

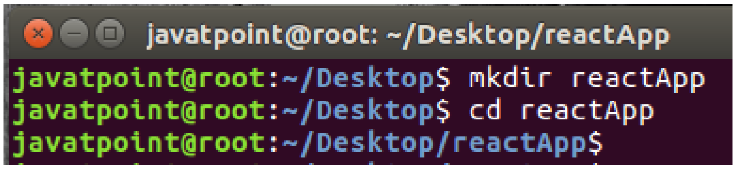
To verify NodeJS and NPM, use the command shown in the below image.



```
javatpoint@root:~$ node -v
v9.11.2
javatpoint@root:~$ npm -v
5.6.0
```

Install React and React DOM

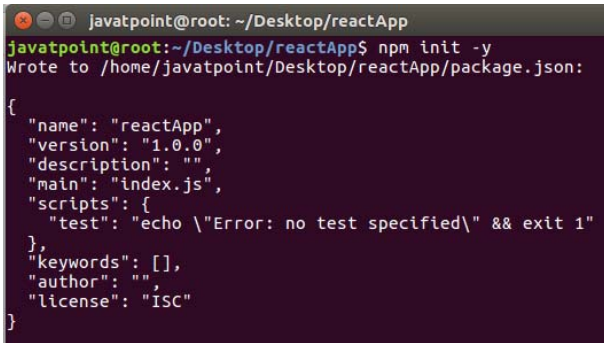
Create a **root** folder with the name **reactApp** on the desktop or where you want. Here, we create it on the desktop. You can create the folder directly or using the command given below.



```
javatpoint@root:~/Desktop$ mkdir reactApp
javatpoint@root:~/Desktop$ cd reactApp
javatpoint@root:~/Desktop/reactApp$
```

Now, you need to create a **package.json** file. To create any module, it is required to generate a package.json file in the project folder. To do this, you need to run the following command as shown in the below image.

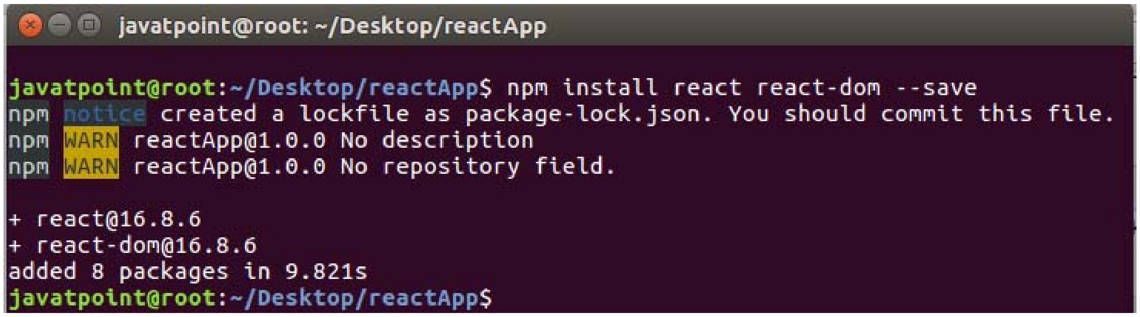
1. javatpoint@root:~/Desktop/reactApp> npm init -y



```
javatpoint@root:~/Desktop/reactApp$ npm init -y
Wrote to /home/javatpoint/Desktop/reactApp/package.json:

{
  "name": "reactApp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

After creating a package.json file, you need to install **react** and its DOM **packages** using the following npm command in the terminal window as shown in the below image.

1. javatpoint@root:~/Desktop/reactApp>npm install react react-dom --save



```
javatpoint@root:~/Desktop/reactApp$ npm install react react-dom --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN reactApp@1.0.0 No description
npm WARN reactApp@1.0.0 No repository field.

+ react@16.8.6
+ react-dom@16.8.6
added 8 packages in 9.821s
javatpoint@root:~/Desktop/reactApp$
```

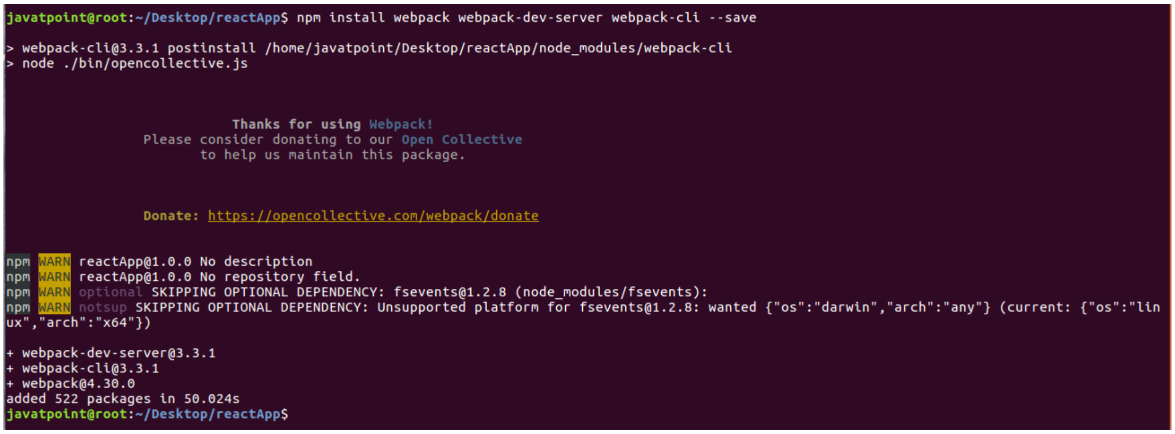
You can also use the above command separately which can be shown as below.

1. javatpoint@root:~/Desktop/reactApp>npm install react --save
2. javatpoint@root:~/Desktop/reactApp>npm install react-dom --save

Install Webpack

Webpack is used for module packaging, development, and production pipeline automation. We will use **webpack-dev-server** during development, **webpack** to create production builds, and **webpack CLI** provides a set of commands. Webpack compiles these into a single file(bundle). To install webpack use the command shown in the below image.

1. javatpoint@root:~/Desktop/reactApp>npm install webpack webpack-dev-server webpack-cli --save



```
javatpoint@root:~/Desktop/reactApp$ npm install webpack webpack-dev-server webpack-cli --save

> webpack-cli@3.3.1 postinstall /home/javatpoint/Desktop/reactApp/node_modules/webpack-cli
> node ./bin/opencollective.js



                  Thanks for using Webpack!
             Please consider donating to our Open Collective
                    to help us maintain this package.



             Donate: https://opencollective.com/webpack/donate


npm WARN reactApp@1.0.0 No description
npm WARN reactApp@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.8 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.8: wanted {"os":"darwin","arch":"any"} (current: {"os":"lin
ux","arch":"x64"})

+ webpack-dev-server@3.3.1
+ webpack-cli@3.3.1
+ webpack@4.30.0
added 522 packages in 50.024s
javatpoint@root:~/Desktop/reactApp$
```

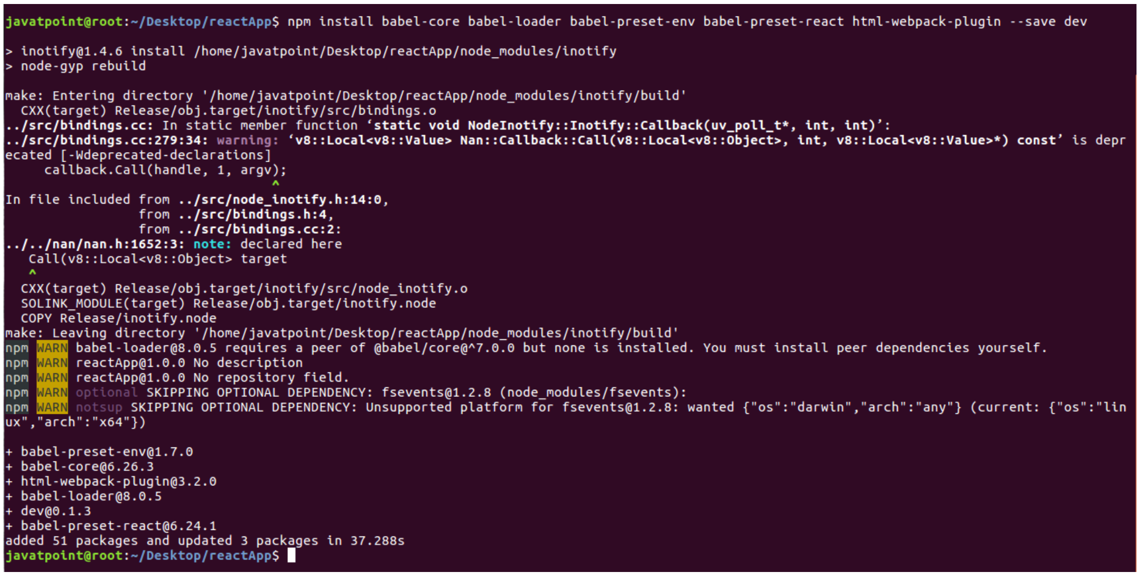
You can also use the above command separately which can be shown as below.

1. javatpoint@root:~/Desktop/reactApp>npm install webpack --save
2. javatpoint@root:~/Desktop/reactApp>npm install webpack-dev-server --save
3. javatpoint@root:~/Desktop/reactApp>npm install webpack-cli --save

Install Babel

Babel is a JavaScript compiler and transpiler used to convert one source code to others. It compiles React JSX and ES6 to ES5 JavaScript which can be run on all browsers. We need **babel-loader** for JSX file types, **babel-preset-react** makes your browser update automatically when any changes occur to your code without losing the current state of the app. ES6 support requires **babel-preset-env** Babel preset. To install webpack use the following command shown in the below image.

1. javatpoint@root:~/Desktop/reactApp>npm install babel-core babel-loader babel-preset-env babel-preset-react babel-webpack-plugin --save-dev



```
javatpoint@root:~/Desktop/reactApp$ npm install babel-core babel-loader babel-preset-env babel-preset-react html-webpack-plugin --save dev

> inotify@1.4.6 install /home/javatpoint/Desktop/reactApp/node_modules/inotify
> node-gyp rebuild

make: Entering directory '/home/javatpoint/Desktop/reactApp/node_modules/inotify/build'
  CXX(target) Release/obj.target/inotify/src/bindings.o
../src/bindings.cc: In static member function ‘static void NodeInotify::Inotify::Callback(uv_poll_t*, int, int)’:
../src/bindings.cc:279:34: warning: ‘v8::Local<v8::Value> Nan::Callback::Call(v8::Local<v8::Object>, int, v8::Local<v8::Value>*) const’ is depr
ecated [-Wdeprecated-declarations]
     callback.Call(handle, 1, argv);
                                  ^
In file included from ../src/node_inotify.h:14:0,
                 from ../src/bindings.h:4,
                 from ../src/bindings.cc:2:
../../nan/nan.h:1652:3: note: declared here
   Call(v8::Local<v8::Object> target
   ^
  CXX(target) Release/obj.target/inotify/src/node_inotify.o
  SOLINK_MODULE(target) Release/obj.target/inotify.node
  COPY Release/inotify.node
make: Leaving directory '/home/javatpoint/Desktop/reactApp/node_modules/inotify/build'
npm WARN babel-loader@8.0.5 requires a peer of @babel/core@^7.0.0 but none is installed. You must install peer dependencies yourself.
npm WARN reactApp@1.0.0 No description
npm WARN reactApp@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.8 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.8: wanted {"os":"darwin","arch":"any"} (current: {"os":"lin
ux","arch":"x64"})

+ babel-preset-env@1.7.0
+ babel-core@6.26.3
+ html-webpack-plugin@3.2.0
+ babel-loader@8.0.5
+ dev@0.1.3
+ babel-preset-react@6.24.1
added 51 packages and updated 3 packages in 37.288s
javatpoint@root:~/Desktop/reactApp$ █
```

You can also use the above command separately which can be shown as below.

1. javatpoint@root:~/Desktop/reactApp>npm install babel-core --save-dev
2. javatpoint@root:~/Desktop/reactApp>npm install babel-loader --save-dev
3. javatpoint@root:~/Desktop/reactApp>npm install babel-preset-env --save-dev
4. javatpoint@root:~/Desktop/reactApp>npm install babel-preset-react --save-dev

```
javatpoint@root:~/Desktop/reactApp>npm install babel-webpack-plugin --save-dev
```

Create Files

To complete the installation process, you need to add the following files in your project folder. These files are **index.html, App.js, main.js, webpack.config.js** and, **.babelrc.** You can create these files by manually, or by using the command prompt.

```
javatpoint@root:~/Desktop/reactApp>touch index.html
javatpoint@root:~/Desktop/reactApp>touch App.js
javatpoint@root:~/Desktop/reactApp>touch main.js
javatpoint@root:~/Desktop/reactApp>touch webpack.config.js
javatpoint@root:~/Desktop/reactApp>touch .babelrc
```

Set Compiler, Loader, and Server for React Application

Configure webpack

You can configure webpack in the **webpack.config.js** file by adding the following code. It defines your app entry point, build output and the extension which will resolve automatically. It also set the development server to **8080** port. It defines the loaders for processing various file types used within your app and wrap up by adding plugins needed during our development.

webpack.config.json

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
    entry: './main.js',
        output: {
            path: path.join(__dirname, '/bundle'),
           filename: 'index_bundle.js'
         },
        devServer: {
               inline: true,
               port: 8080
         },
         module: {
```

```
  rules: [
     {
  test: /\.jsx?$/,
     exclude: /node_modules/,
     use: {
         loader: "babel-loader",
       }
      }
  ]
 },
  plugins:[
  new HtmlWebpackPlugin({
  template: './index.html'
   })
  ]
}
```

Now, open the **package.json** file and delete ***"test" "echo \" Error: no test specified\" && exit 1"*** inside ***"scripts"*** object, then add the start and build commands instead. It is because we will not perform any testing in this app.

```
{
  "name": "reactApp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "webpack-dev-server --mode development --open --hot",
    "build": "webpack --mode production"
  },
  "keywords": [],
          "author": "",
  "license": "ISC",
          "dependencies": {
    "react": "^16.8.6",
            "react-dom": "^16.8.6",
    "webpack-cli": "^3.3.1",
            "webpack-dev-server": "^3.3.1"
  },
          "devDependencies": {
    "@babel/core": "^7.4.3",
            "@babel/preset-env": "^7.4.3",
    "@babel/preset-react": "^7.0.0",
            "babel-core": "^6.26.3",
   "babel-loader": "^8.0.5",
            "babel-preset-env": "^1.7.0",
   "babel-preset-react": "^6.24.1",
            "html-webpack-plugin": "^3.2.0",
   "webpack": "^4.30.0"
         }
}
```

HTML webpack template for index.html

We can add a custom template to generate **index.html** using the **HtmlWebpackPlugin** plugin. This enables us to add a viewport tag to support mobile responsive scaling of our app. It also set the **div id = "app"** as a root element for your app and adding the **index_bundle.js** script, which is our bundled app file.

```
<!DOCTYPE html>
   <html lang = "en">
      <head>
         <meta charset = "UTF-8">
         <title>React App</title>
      </head>
      <body>
         <div id = "app"></div>
         <script src = 'index_bundle.js'></script>
      </body>
</html>
```

App.jsx and main.js

This is the first React component, i.e. app entry point. It will render Hello World.

App.js

```
import React, { Component } from 'react';
class App extends Component{
   render(){
      return(
         <div>
            <h1>Hello World</h1>
         </div>
      );
   }
}
```

```
export default App;
```

Now, import this component and render it to your root App element so that you can see it in the browser.

Main.js

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

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

> ***Note: If you want to use something, you need to import it first. To make the component usable in other parts of the app, you need to export it after creation and import it in the file where you want to use it.***

Create .babelrc file

Create a file with name **.babelrc** and copy the following code to it.

.babelrc

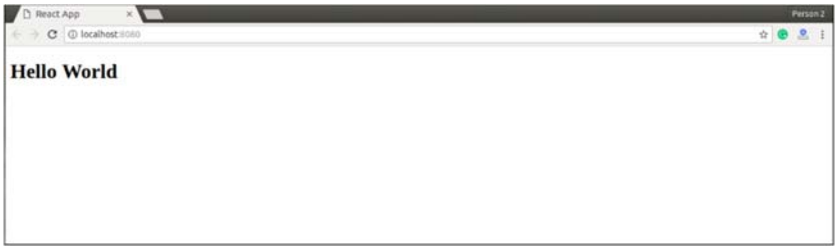
```
{
   "presets":[
  "@babel/preset-env", "@babel/preset-react"]
}
```

Running the Server

After completing the installation process and setting up the app, you can start the server by running the following command.

```
javatpoint@root:~/Desktop/reactApp>npm start
```

It will show the port number which we need to open in the browser. After we open it, you will see the following output.



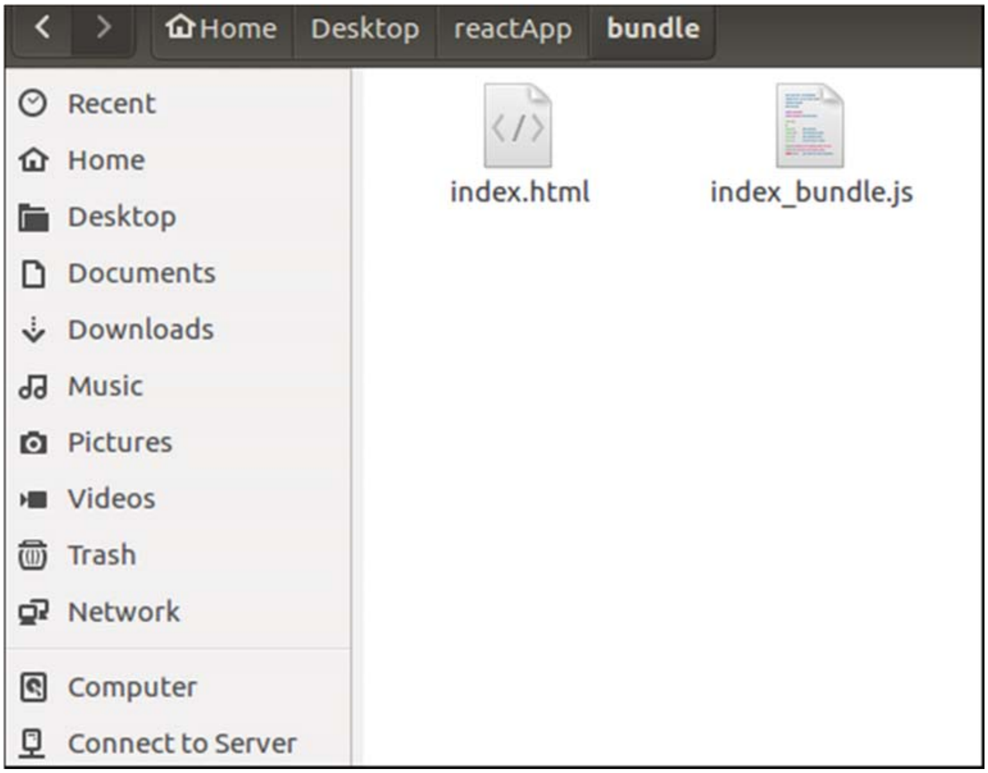
Hello World

Generate the Bundle

Now, generate the bundle for your app. Bundling is the process of following imported files and merging them into a single file: a **"bundle."** This bundle can then be included on a webpage to load an entire app at once. To generate this, you need to run the build command in command prompt which can be shown below.

```
1. javatpoint@root:~/Desktop/reactApp> npm run build
```

This command will generate the bundle in the current folder(in which your app belongs) and will be shown as like below image.



2. Using the create-react-app command

If you do not want to install react by using webpack and babel, then you can choose create-react-app to install react. The 'create-react-app' is a tool maintained by Facebook itself. This is suitable for beginners without manually having to deal with transpiling tools like webpack and babel. In this section, I will be showing you how to install React using CRA tool.

Install NodeJS and NPM

NodeJS and NPM are the platforms need to develop any ReactJS application. You can install NodeJS and NPM package manager by the link given below.

https://www.javatpoint.com/install-nodejs-on-linux-ubuntu-centos

Install React

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.

```
javatpoint@root:~/>npm install -g create-react-app
```

Create a new React project

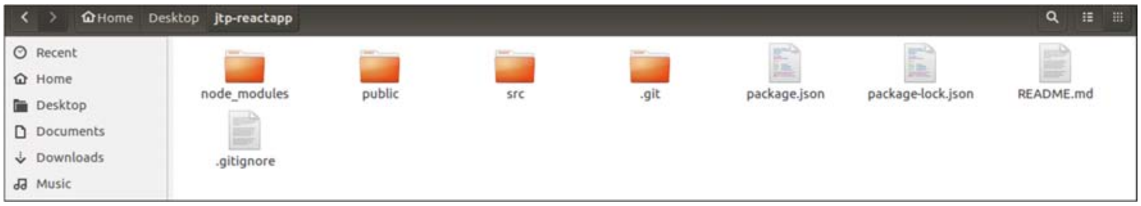
After the installation of React, you can create a new react project using create-react-app command. Here, I choose **jtp-reactapp** name for my project.

```
javatpoint@root:~/>create-react-app jtp-reactapp
```

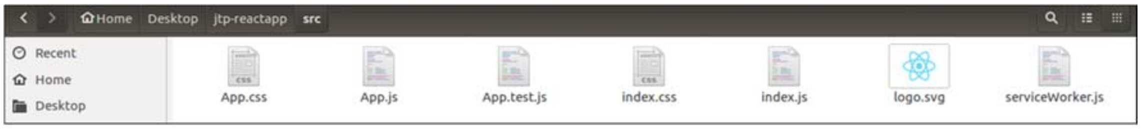
NOTE: You can combine the above two steps in a single command using npx. The npx is a package runner tool that comes with npm 5.2 and above version.

```
javatpoint@root:~/>npx create-react-app jtp-reactapp
```

The above command will install the react and create a new project with the name jtp-reactapp. 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 JavaTpoint.

            <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;
```

NOTE: ***You can also choose your own favorite code editor for editing your project. But in my case, I choose Eclipse. Using the below link, you can download Eclipse for Ubuntu and install.***

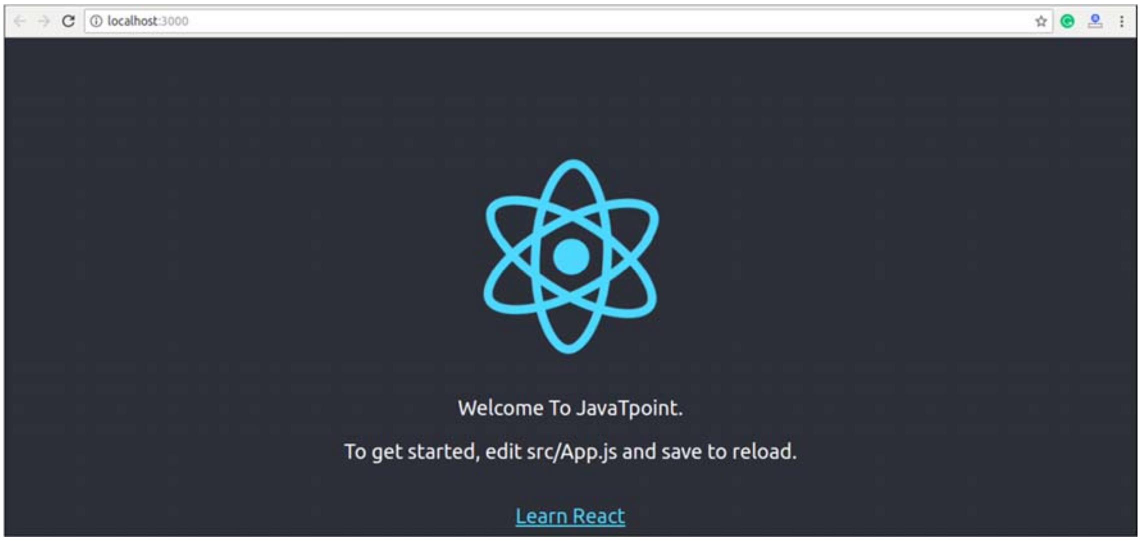
click Here to download Eclipse for Ubuntu and install

Running the Server

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

```
1. javatpoint@root:~/Desktop>cd jtp-reactapp
2. javatpoint@root:~/Desktop/jtp-reactapp>npm start
```

It will show the port number which we need to open in the browser. After we open it, you will see the following output.



Welcome To JavaTpoint.

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

5. React create-react-app

Starting a new React project is very complicated, with so many build tools. It uses many dependencies, configuration files, and other requirements such as Babel, Webpack, ESLint before writing a single line of React code. Create React App CLI tool removes all that complexities and makes React app simple. For this, you need to install the package using NPM, and then run a few simple commands to get a new React project.

The **create-react-app** is an excellent tool for beginners, which allows you to create and run React project very quickly. It does not take any configuration manually. This tool is wrapping all of the required dependencies like **Webpack**, **Babel** for React project itself and then you need to focus on writing React code only. This tool sets up the development environment, provides an excellent developer experience, and optimizes the app for production.

Requirements

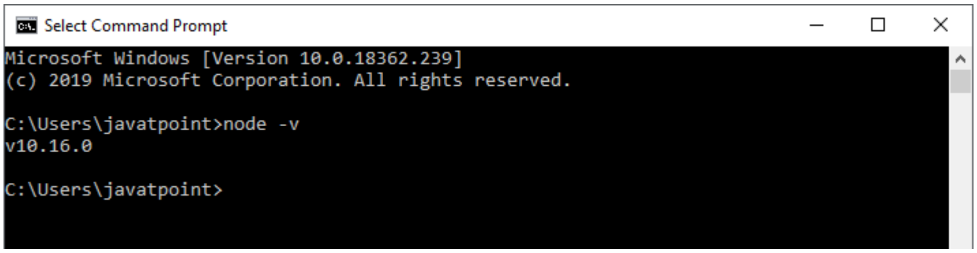
The Create React App is maintained by **Facebook** and can works on any **platform**, for example, macOS, Windows, Linux, etc. To create a React Project using create-react-app, you need to have installed the following things in your system.

1. Node version >= 8.10
2. NPM version >= 5.6

Let us check the current version of **Node** and **NPM** in the system.

Run the following command to check the Node version in the command prompt.

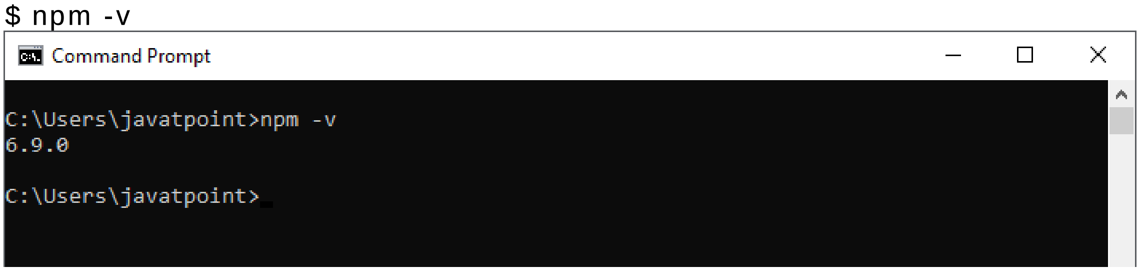
1. $ node -v



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\javatpoint>node -v
v10.16.0

C:\Users\javatpoint>
```

Run the following command to check the NPM version in the command prompt.

1. $ npm -v



```
C:\Users\javatpoint>npm -v
6.9.0

C:\Users\javatpoint>_
```

Installation

Here, we are going to learn how we can install React using ***CRA*** tool. For this, we need to follow the steps as given below.

Install React

We can install React using npm package manager by using the following command. There is no need to worry about the complexity of React installation. The create-react-app npm package manager will manage everything, which needed for React project.

1. C:\Users\javatpoint> npm install -g create-react-app

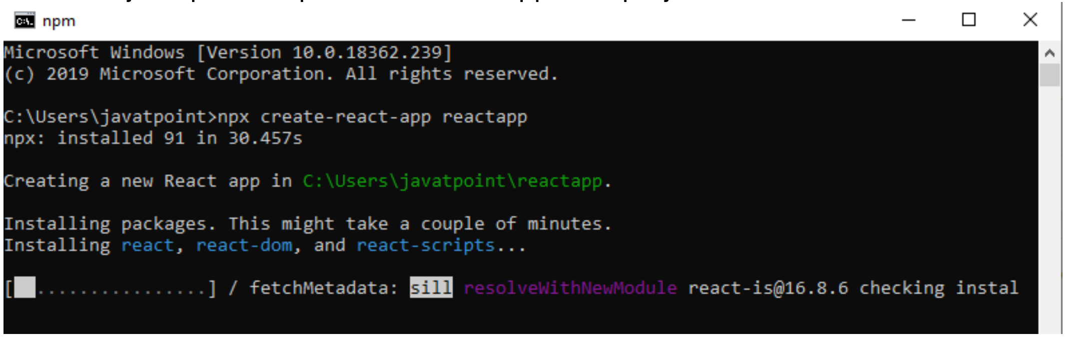
Create a new React project

Once the React installation is successful, we can create a new React project using create-react-app command. Here, I choose "reactproject" name for my project.

1. C:\Users\javatpoint> create-react-app reactproject

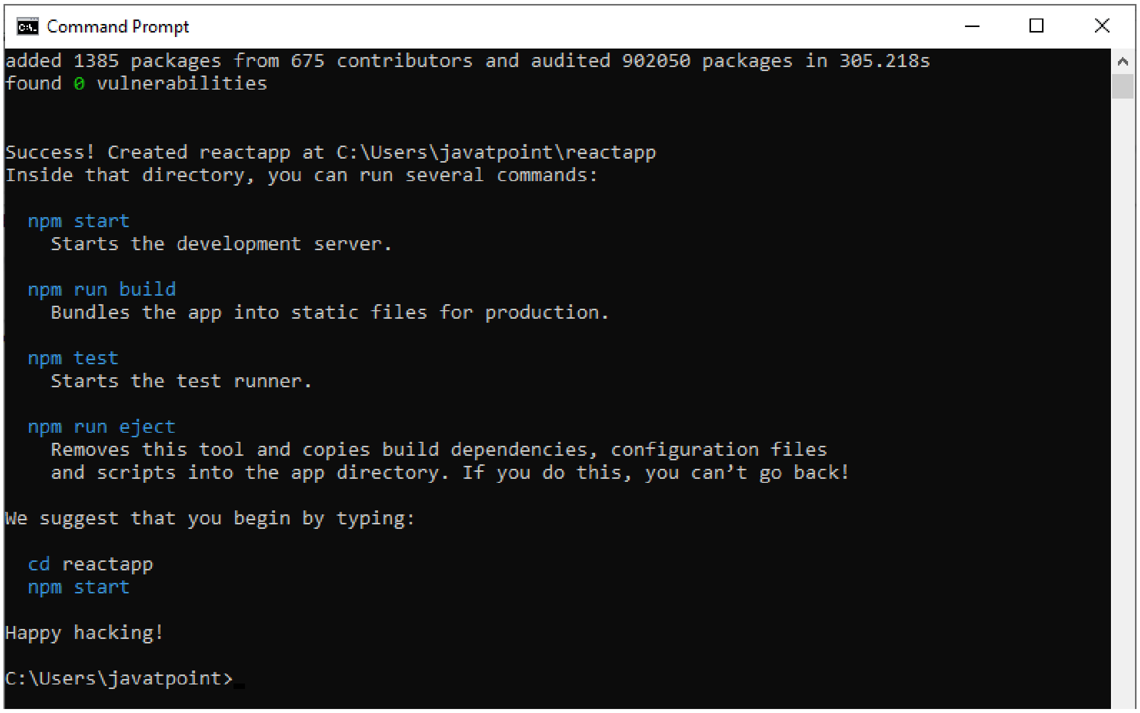
NOTE: ***We can combine the above two steps in a single command using*** *npx*. ***The npx is a package runner tool which comes with npm 5.2 and above version.***

1. C:\Users\javatpoint> npx create-react-app reactproject



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\javatpoint>npx create-react-app reactapp
npx: installed 91 in 30.457s

Creating a new React app in C:\Users\javatpoint\reactapp.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

[█...............] / fetchMetadata: sill resolveWithNewModule react-is@16.8.6 checking instal
```

The above command will take some time to install the React and create a new project with the name "reactproject." Now, we can see the terminal as like below.

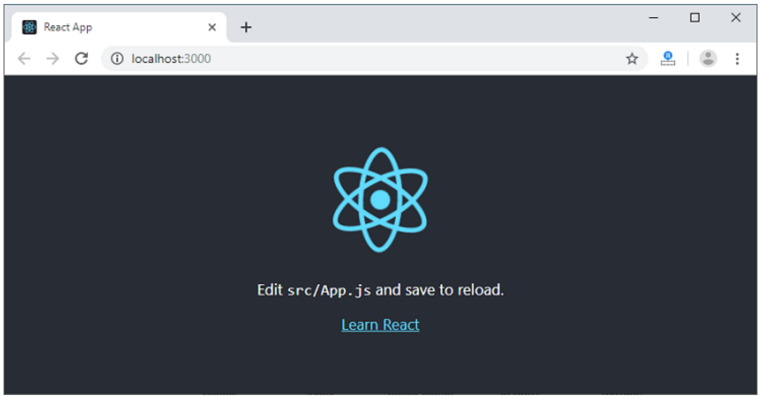


```
added 1385 packages from 675 contributors and audited 902050 packages in 305.218s
found 0 vulnerabilities


Success! Created reactapp at C:\Users\javatpoint\reactapp
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

  cd reactapp
  npm start

Happy hacking!

C:\Users\javatpoint>_
```

The above screen tells that the React project is created successfully on our system. Now, we need to start the server so that we can access the application on the browser. Type the following command in the terminal window.

1. $ cd Desktop
2. $ npm start

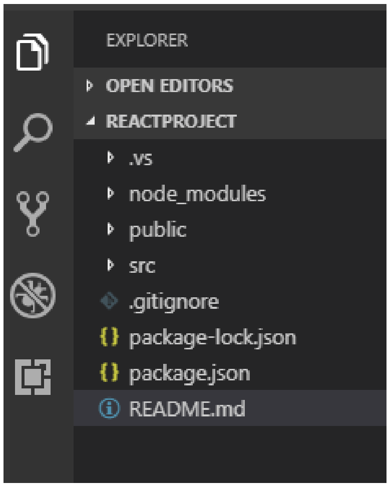
NPM is a package manager which starts the server and access the application at default server http://localhost:3000. Now, we will get the following screen.



Edit `src/App.js` and save to reload.

Learn React

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



EXPLORER
OPEN EDITORS
REACTPROJECT
.vs
node_modules
public
src
.gitignore
package-lock.json
package.json
README.md

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

1. **node_modules:** It contains the React library and any other third-party libraries needed.
2. **public:** It holds the public assets of the application. It contains the index.html where React will mount the application by default on the <div id="root"></div> element.
3. **src:** It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files. Here, the App.js file always responsible for displaying the output screen in React.
4. **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.
5. **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.
6. **README.md:** It provides the documentation to read about React topics.

React Environment Setup

Now, open the **src >> App.js** file and make changes which you want to display on the screen. After making desired changes, **save** the file. As soon as we save the file, Webpack recompiles the code, and the page will refresh automatically, and changes are reflected on the browser screen. Now, we can create as many components as we want, import the newly created component inside the **App.js** file and that file will be included in our main **index.html** file after compiling by Webpack.

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.

1. $ npm build

6. React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem. The important features of ReactJS are as following.

- JSX
- Components
- One-way Data Binding
- Virtual DOM
- Simplicity
- Performance

JSX

JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML like syntax used by ReactJS. This syntax is processed into JavaScript calls of React Framework. It extends the ES6 so that HTML like text can co-exist with JavaScript react code. It is not necessary to use JSX, but it is recommended to use in ReactJS.

Components

ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls. These components can be reusable which help you to maintain the code when working on larger scale projects.

One-way Data Binding

ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control throughout the application. If the data flow is in another direction, then it requires additional features. It is because components are supposed to be immutable and the data within them cannot be changed. Flux is a pattern that helps to keep your data unidirectional. This makes the application more flexible that leads to increase efficiency.

Virtual DOM

A virtual DOM object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

Simplicity

ReactJS uses JSX file which makes the application simple and to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.

Performance

ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM. Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

7. Pros and Cons of ReactJS

Today, ReactJS is the highly used open-source JavaScript Library. It helps in creating impressive web apps that require minimal effort and coding. The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. There are important pros and cons of ReactJS given as following:

Advantage of ReactJS

1. Easy to Learn and USe

ReactJS is much easier to learn and use. It comes with a good supply of documentation, tutorials, and training resources. Any developer who comes from a JavaScript background can easily understand and start creating web apps using React in a few days. It is the V(view part) in the MVC (Model-View-Controller) model, and referred to as ?one of the JavaScript frameworks.? It is not fully featured but has the advantage of open-source JavaScript User Interface(UI) library, which helps to execute the task in a better manner.

2. Creating Dynamic Web Applications Becomes Easier

To create a dynamic web application specifically with HTML strings was tricky because it requires a complex coding, but React JS solved that issue and makes it easier. It provides less coding and gives more functionality. It makes use of the JSX(JavaScript Extension), which is a particular syntax letting HTML quotes and HTML tag syntax to render particular subcomponents. It also supports the building of machine-readable codes.

3. Reusable Components

A ReactJS web application is made up of multiple components, and each component has its own logic and controls. These components are responsible for outputting a small, reusable piece of HTML code which can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain. 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.

4. Performance Enhancement

ReactJS improves performance due to virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. Most of the developers faced the problem when the DOM was updated, which slowed down the performance of the application. ReactJS solved this problem by introducing virtual 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, leading to smoother and faster performance.

5. The Support of Handy Tools

React JS has also gained popularity due to the presence of a handy set of tools. These tools make the task of the developers understandable and easier. The React Developer Tools have been designed as Chrome and Firefox dev extension and allow you to inspect the React component hierarchies in the virtual DOM. It also allows you to select particular components and examine and edit their current props and state.

6. Known to be SEO Friendly

Traditional JavaScript frameworks have an issue in dealing with SEO. The search engines generally having trouble in reading JavaScript-heavy applications. Many web developers have often complained about this problem. ReactJS overcomes this problem that helps developers to be easily navigated on various search engines. It is because React.js applications can run on the server, and the virtual DOM will be rendering and returning to the browser as a regular web page.

7. The Benefit of Having JavaScript Library

Today, ReactJS is choosing by most of the web developers. It is because it is offering a very rich JavaScript library. The JavaScript library provides more flexibility to the web developers to choose the way they want.

8. Scope for Testing the Codes

ReactJS applications are extremely easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

Disadvantage of ReactJS

1. The high pace of development

The high pace of development has an advantage and disadvantage both. In case of disadvantage, since the environment continually changes so fast, some of the developers not feeling comfortable to relearn the new ways of doing things regularly. It may be hard for them to adopt all these changes with all the continuous updates. They need to be always updated with their skills and learn new ways of doing things.

2. Poor Documentation

It is another cons which are common for constantly updating technologies. React technologies updating and accelerating so fast that there is no time to make proper documentation. To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.

3. View Part

ReactJS Covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

4. JSX as a barrier

ReactJS uses JSX. It's a syntax extension that allows HTML with JavaScript mixed together. This approach has its own benefits, but some members of the development community consider JSX as a barrier, especially for new developers. Developers complain about its complexity in the learning curve.

Unit-8

Difference Between AngularJS and ReactJS

AngularJS

AngularJS is an open-source JavaScript framework used to build a dynamic web application. Misko Hevery and Adam Abrons developed AngularJS in 2009, and now Google maintained it. The latest version of Angular is 1.7.8 on March 11, 2019. It is based on HTML and JavaScript and mostly used for building a Single Page Application. It can be included to an HTML page with a <script> tag. It extends HTML by adding built-in attributes with the directive and binds data to HTML with Expressions.

Features of AngularJS

1. **Data-binding:** AngularJS follows the two-way data binding. It is the automatic synchronization of data between model and view components.
2. **POJO Model:** AngularJS uses POJO (Plain Old JavaScript) model, which provides spontaneous and well-planned objects. The POJO model makes AngularJS self-sufficient and easy to use.
3. **Model View Controller(MVC) Framework:** MVC is a software design pattern used for developing web applications. The working model of AngularJS is based on MVC patterns. The MVC Architecture in AngularJS is easy, versatile, and dynamic. MVC makes it easier to build a separate client-side application.
4. **Services:** AngularJS has several built-in services such as $http to make an XMLHttpRequest.
5. **User interface with HTML:** In AngularJS, User interfaces are built on HTML. It is a declarative language which has shorter tags and easy to comprehend. It provides an organized, smooth, and structured interface.
6. **Dependency Injection:** AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

7. **Active community on Google:** AngularJS provides excellent community support. It is Because Google maintains AngularJS. So, if you have any maintenance issues, there are many forums available where you can get your queries solved.
8. **Routing:** Routing is the transition from one view to another view. Routing is the key aspect of single page applications where everything comes in a single page. Here, developers do not want to redirect the users to a new page every time they click the menu. The developers want the content load on the same page with the URL changing.

ReactJS

ReactJS is an open-source JavaScript library used to build a user interface for Single Page Application. It is responsible only for the view layer of the application. It provides developers to compose complex UIs from a small and isolated piece of code called "components." ReactJS made of two parts first is components, that are the pieces that contain HTML code and what you want to see in the user interface, and the second one is HTML document where all your components will be rendered.

Jordan Walke, who was a software engineer at Facebook, develops it. Initially, it was developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 for the newsfeed section, but it was released to the public in May 2013.

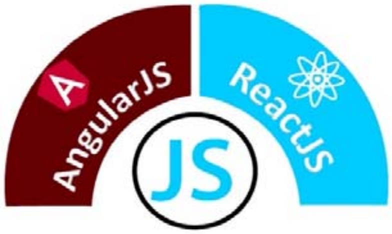
Features of ReactJS

1. **JSX:** JSX is a JavaScript syntax extension. The JSX syntax is processed into JavaScript calls of React Framework. It extends the ES6 so that HTML like text can co-exist with JavaScript React code.
2. **Components:** ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its logic and controls. These components can be reusable, which help you to maintain the code when working on larger scale projects.
3. **One-way Data Binding:** ReactJS follows unidirectional data flow or one-way data binding. The one-way data binding gives you better control throughout the application. If the data flow is in another direction, then it

requires additional features. It is because components are supposed to be immutable, and the data within them cannot be changed.

4. **Virtual DOM:** A virtual DOM object is a representation of the real DOM object. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then, it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that are changed. It makes the application faster, and there is no wastage of memory.
5. **Simplicity:** ReactJS uses the JSX file, which makes the application simple and to code as well as understand. Also, ReactJS is a component-based approach which makes the code reusable as your need. It makes it simple to use and learn.
6. **Performance:** ReactJS is known to be a great performer. The reason behind this is that it manages a virtual DOM. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM. Instead, we are writing virtual Components that will turn into the DOM, leading to smoother and faster performance.

AngularJS Vs. ReactJS



	AngularJS	ReactJS
Author	Google	Facebook Community
Developer	Misko Hevery	Jordan Walke
Initial Release	October 2010	March 2013

Latest Version	Angular 1.7.8 on 11 March 2019.	React 16.8.6 on 27 March 2019
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open Source JS Framework
Rendering	Client-Side	Server-Side
Packaging	Weak	Strong
Data-Binding	Bi-directional	Uni-directional
DOM	Regular DOM	Virtual DOM
Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux
Dependencies	It manages dependencies automatically.	It requires additional tools to manage dependencies.
Routing	It requires a template or controller to its router configuration, which has to be managed manually.	It doesn't handle routing but has a lot of modules for routing, eg., react-router.
Performance	Slow	Fast, due to virtual DOM.
Best For	It is best for single page applications that update a single view at a time.	It is best for single page applications that update multiple views at a time.

Unit-9

Difference between ReactJS and React Native

ReactJS

ReactJS is an open-source JavaScript library used to build the user interface for Web Applications. It is responsible only for the view layer of the application. It provides developers to compose complex UIs from a small and isolated piece of code called "components." ReactJS made of two parts first is components, that are the pieces that contain HTML code and what you want to see in the user interface, and the second one is HTML document where all your components will be rendered.

Jordan Walke, who was a software engineer at Facebook, develops it. Initially, it was developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 for the newsfeed section, but it was released to the public in May 2013.

Advantage of ReactJS

1. **Easy to Learn and Use:** ReactJS is much easier to learn and use. Any developer who comes from a JavaScript background can easily understand and start creating web apps using React.
2. **Creating Dynamic Web Applications Becomes Easier:** To create a dynamic web application specifically with HTML was tricky, which requires complex coding, but React JS solved that issue and makes it easier. It provides less coding and gives more functionality.
3. **Reusable Components:** A ReactJS web application is made up of multiple components, and each component has its logic and controls. These components can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain.
4. **Performance Enhancement:** ReactJS improves performance due to virtual 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, leading to smoother and faster performance.

5. **The Support of Handy Tools:** ReactJS support a handy set of tools which make the task of the developers understandable and easier. It also allows you to select particular components and examine and edit their current Props and State.
6. **Known to be SEO Friendly:** Traditional JavaScript frameworks have an issue in dealing with SEO. ReactJS overcomes this problem, which helps developers to be easily navigated on various search engines. It is because ReactJS applications can run on the server, and the virtual DOM will be rendering and returning to the browser as a regular web page.
7. **The Benefit of Having JavaScript Library:** Today, ReactJS gaining popularity among web developers. It is offering a very rich JavaScript library which provides more flexibility to the web developers to choose the way they want.
8. **Scope for Testing the Codes:** ReactJS applications are easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

Disadvantage of ReactJS

1. **The high pace of development:** As we know, the frameworks continually changes so fast. The developers are not feeling comfortable to re-learn the new ways of doing things regularly. It may be hard for them to adopt all these changes with all the continuous updates.
2. **Poor Documentation:** React technologies updating and accelerating so fast that there is no time to make proper documentation. To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.
3. **View Part:** ReactJS covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

React Native

React Native is an open-source JavaScript framework used for developing a mobile application for iOS Android, and Windows. It uses only JavaScript to build a cross-platform mobile app. React Native is same as React, but it uses native components instead of using web components as building blocks. It targets mobile platforms rather than the browser.

Unit-10

React vs. Vue

React and Vue is the two most popular JavaScript libraries which are used to build thousands of websites today. Both React and Vue are very powerful frameworks with their own set of pros and cons. Which one you have to pick, depends on the business needs and use cases.

Both React and Vue have a lot of common things like the component-based architecture, usage of virtual DOM, usage of props, chrome Dev tools for debugging, and many more. But, both have some significant differences, which are given below.

	React	Vue
Definition	React is a declarative, efficient, flexible, open-source JavaScript library for building reusable UI components.	Vue is an open-source JavaScript library for building reusable user interfaces and single-page applications.

History	It was created by Jordan Walke, a software engineer at Facebook. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram. Facebook developed React in 2011 for the newsfeed section, but it was released to the public on May 2013.	Vue was created by Evan You, a former employee of Google worked on many Angular projects. He wanted to make a better version of Angular, just extracting the part which he liked about Angular and making it lighter. The first release of Vue was introduced in February 2014.
Learning Curve	React is not a complete framework, and the more advanced framework must be looked for the use of third-party libraries. It makes the learning of the core framework not so easy. It adds some complexity to the learning curve as it differs based on the choices you take with additional functionality.	Vue provides higher customizability, which makes it easier to learn than Angular or React. Vue shares some concepts with Angular and React in their functionality. Hence, the transition to Vue from Angular and React is an easy option. Also, the official documentation is well written and covers everything the developer needs to build a Vue app.
Preferred Language	JavaScript/JavaScript XML	HTML/JavaScript
Size	The size of the React library is 100 kilobytes (approx.).	The size of the Vue library is 60 kilobytes (approx.).
Performance	Its performance is slow as compared to Vue.	Its performance is fast as compared to React.
Flexibility	React provides great flexibility to support third-party libraries.	Vue provides limited flexibility as compared to React.

Coding Style	React uses JSX for writing JavaScript Expression instead of regular JavaScript. JSX is similar to HTML code within the JavaScript expressions. React takes everything as Component, and each component has its own lifecycle methods.	Vue coding style is little similar to Angular. It separates HTML, JS, and CSS as like web developers have been used to the web development scenario for years. But, it also allows using JSX if you prefer. Vue's take of the component lifecycle more intuitive than React's.
Data Binding	React supports one-way data binding. The one-way data binding refers to a single source of truth. React flows in a single direction, and only the model can change the app's state.	Vue supports both one-way and two-way data binding. The two-way data binding is a mechanism where UI fields are bound to model dynamically. If the UI components change, model data is also changed accordingly.
Tooling	React has great tooling support. It uses third-party CLI tool (create-react-app), which helps to create new apps and components in React Project. It has excellent support for the major IDEs.	Vue provides limited tooling support as compared to React. It has a Vue CLI tool, which is similar to the create-react-app tool. It gives supports for major IDEs but not as good as React.
Current Version	React 16.8.6 on March 27, 2019	Vue 2.6.10 on March 20, 2019.
Long Term Support	It is suitable for long term supports.	It is not suitable for long term support.

Unit-11

React 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.

Example

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

JSX File

```
<div>Hello JavaTpoint</div>
```

Corresponding Output

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

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 JavaTpoint."

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.

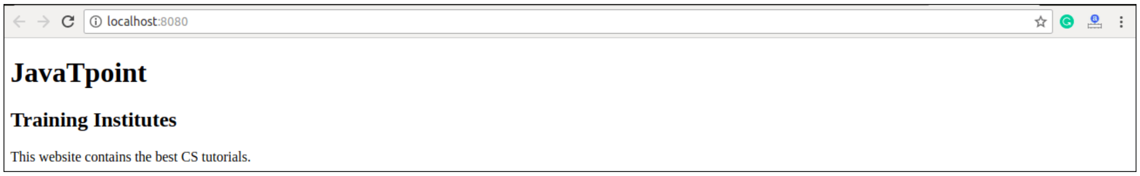
Nested Elements in JSX

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.

App.JSX

```
import React, { Component } from 'react';
class App extends Component{
   render(){
      return(
         <div>
            <h1>JavaTpoint</h1>
          <h2>Training Institutes</h2>
            <p>This website contains the best CS tutorials.</p>
         </div>
      );
   }
}
export default App;
```

Output:



JavaTpoint

Training Institutes

This website contains the best CS tutorials.

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.

Example

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
         <h1>JavaTpoint</h1>
        <h2>Training Institutes</h2>
         <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

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 JavaTpoint</h2>;
```

Example

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1 className = "hello" >JavaTpoint</h1>
        <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

Output:

```
JavaTpoint
This website contains the best CS tutorials.
```

2. As Expressions: We can specify the values of attributes as expressions using curly braces {}:

```
var element = <h2 className = {varName}>Hello JavaTpoint</h2>;
```

Example

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

Output:

```
45
```

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 JavaTpoint</h1>
```

```
            {/* This is a comment in JSX */}
         </div>
      );
   }
}
export default App;
```

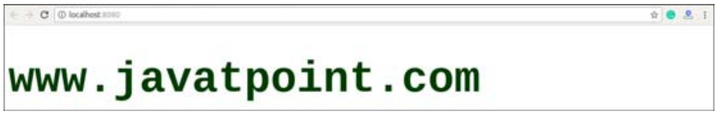
JSX Styling

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.javatpoint.com</h1>
         </div>
      );
   }
}
export default App;
```

Output:



```
www.javatpoint.com
```

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;
```

Output:

```
False!
```

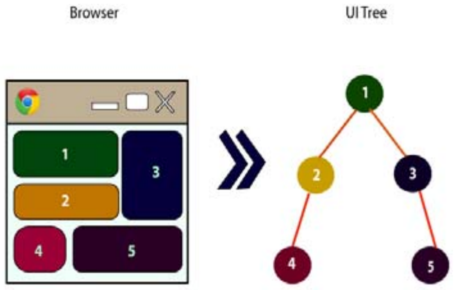
Unit-12

React 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.

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.

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.



In ReactJS, we have mainly two types of components. They are

1. Functional Components
2. Class Components

Functional Components

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.

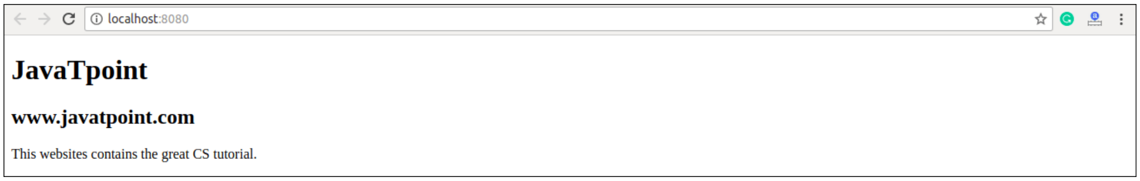
```
function WelcomeMessage(props) {
  return <h1>Welcome to the , {props.name}</h1>;
}
```

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.

Example

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
            <First/>
            <Second/>
         </div>
      );
   }
}
class First extends React.Component {
  render() {
    return (
      <div>
        <h1>JavaTpoint</h1>
      </div>
    );
  }
}
class Second extends React.Component {
  render() {
    return (
      <div>
        <h2>www.javatpoint.com</h2>
        <p>This websites contains the great CS tutorial.</p>
      </div>
    );
  }
}
export default App;
```

Output:



```
JavaTpoint

www.javatpoint.com

This websites contains the great CS tutorial.
```

Class Components

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>
    );
  }
}
```

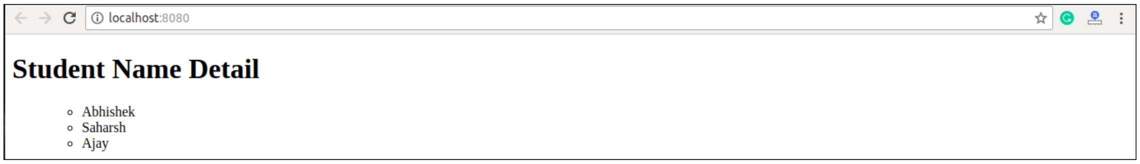
The class component is also known as a stateful component because they can hold or manage local state. It can be explained in the below example.

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;
```

Output:



localhost:8080

Student Name Detail

- Abhishek
- Saharsh
- Ajay

Unit-13

React State

The state is an updatable structure that is used to contain data or information about the component. The state in a component can change over time. The change in state over time can happen as a response to user action or system event. A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.

A state must be kept as simple as possible. It can be set by using the **setState()** method and calling setState() method triggers UI updates. A state represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly. To set an initial state before any interaction occurs, we need to use the **getInitialState()** method.

For example, if we have five components that need data or information from the state, then we need to create one container component that will keep the state for all of them.

Defining State

To define a state, you have to first declare a default set of values for defining the component's initial state. To do this, add a class constructor which assigns an initial state using this.state. The '**this.state**' property can be rendered inside **render()** method.

Example

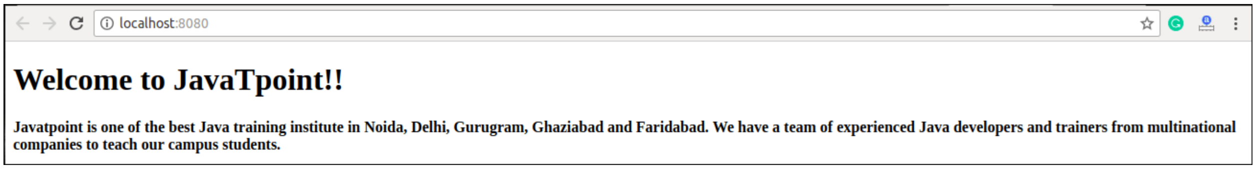
The below sample code shows how we can create a stateful component using ES6 syntax.

```
import React, { Component } from 'react';
class App extends React.Component {
 constructor() {
      super();
      this.state = { displayBio: true };
      }
      render() {
          const bio = this.state.displayBio ? (
              <div>
                  <p><h3>Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java developers and trainers from multinational companies to teach our campus students.</h3></p>
              </div>
              ) : null;
              return (
                  <div>
                      <h1> Welcome to JavaTpoint!! </h1>
                      { bio }
                  </div>
              );
     }
}
```

```
export default App;
```

To set the state, it is required to call the super() method in the constructor. It is because this.state is uninitialized before the super() method has been called.

Output



Welcome to JavaTpoint!!

Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java developers and trainers from multinational companies to teach our campus students.

Changing the State

We can change the component state by using the setState() method and passing a new state object as the argument. Now, create a new method toggleDisplayBio() in the above example and bind this keyword to the toggleDisplayBio() method otherwise we can't access this inside toggleDisplayBio() method.

```
this.toggleDisplayBio = this.toggleDisplayBio.bind(this);
```

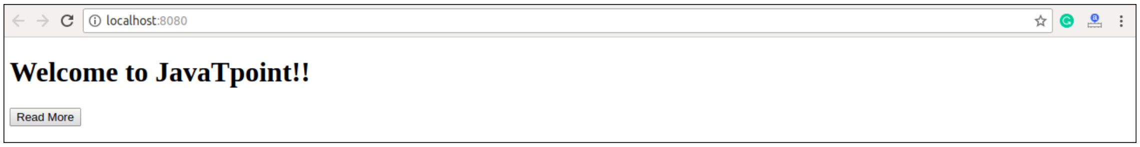
Example

In this example, we are going to add a **button** to the **render**() method. Clicking on this button triggers the toggleDisplayBio() method which displays the desired output.

```
import React, { Component } from 'react';
class App extends React.Component {
 constructor() {
     super();
     this.state = { displayBio: false };
     console.log('Component this', this);
     this.toggleDisplayBio = this.toggleDisplayBio.bind(this);
     }
     toggleDisplayBio(){
         this.setState({displayBio: !this.state.displayBio});
         }
     render() {
         return (
             <div>
                 <h1>Welcome to JavaTpoint!!</h1>
```

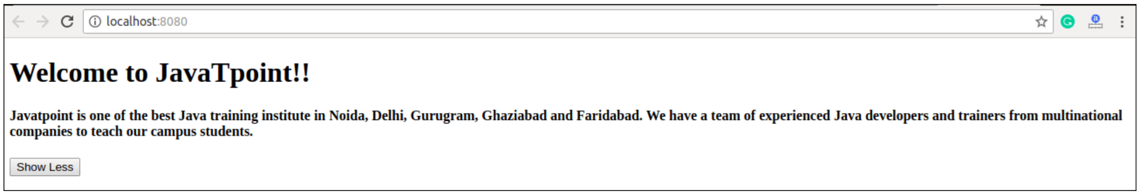
```
        {
          this.state.displayBio ? (
            <div>
              <p><h4>Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java developers and trainers from multinational companies to teach our campus students.</h4></p>
              <button onClick={this.toggleDisplayBio}> Show Less </button>
           </div>
            ) : (
              <div>
                <button onClick={this.toggleDisplayBio}> Read More </button>
              </div>
            )
        }
      </div>
    )
  }
}
export default App;
```

Output:



Welcome to JavaTpoint!!

When you click the **Read More** button, you will get the below output, and when you click the **Show Less** button, you will get the output as shown in the above image.



Welcome to JavaTpoint!!

Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java developers and trainers from multinational companies to teach our campus students.

Unit-14

React 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. It can be explained in the below example.

Example

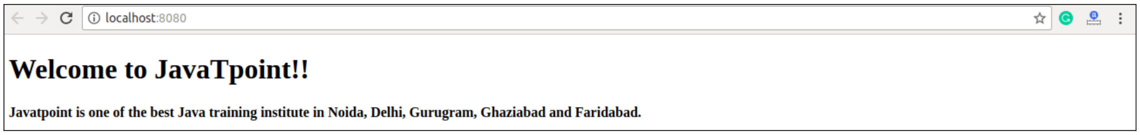
App.js

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
           <h1> Welcome to { this.props.name } </h1>
           <p> <h4> Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. </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 = "JavaTpoint!!" />, document.getElementById('app'));
```

Output



Welcome to JavaTpoint!!

Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad.

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. It can be explained in the below example.

Example

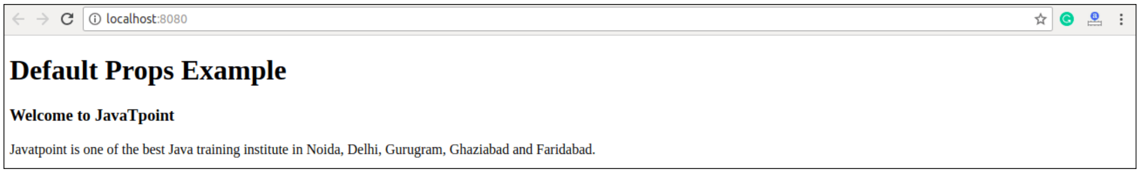
App.js

```
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>Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad.</p>
          </div>
        );
    }
}
App.defaultProps = {
   name: "JavaTpoint"
}
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'));
```

Output



Default Props Example

Welcome to JavaTpoint

Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad.

State and Props

It is possible to combine both state and props in your app. You can set the state in the parent component and pass it in the child component using props. It can be shown in the below example.

Example

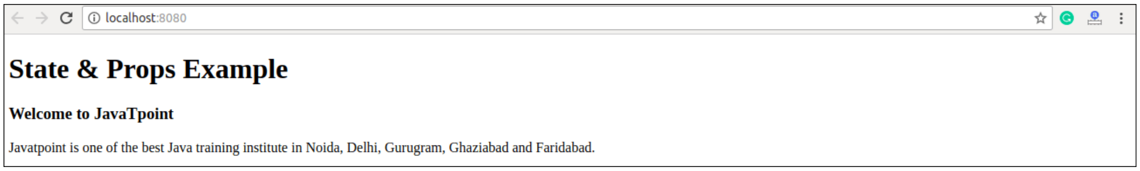
App.js

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "JavaTpoint",
    }
  }
  render() {
    return (
      <div>
        <JTP jtpProp = {this.state.name}/>
      </div>
    );
  }
}
class JTP extends React.Component {
  render() {
    return (
        <div>
            <h1>State & Props Example</h1>
            <h3>Welcome to {this.props.jtpProp}</h3>
            <p>Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad.</p>
        </div>
    );
  }
}
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'));
```

Output:



State & Props Example

Welcome to JavaTpoint

Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad.

Unit-15

React Props Validation

Props are an important mechanism for passing the **read-only** attributes to React components. The props are usually required to use correctly in the component. If it is not used correctly, the components may not behave as expected. Hence, it is required to use **props validation** in improving react components.

Props validation is a tool that will help the developers to avoid future bugs and problems. It is a useful way to force the correct usage of your components. It makes your code more readable. React components used special property **PropTypes** that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes. However, if you use propTypes with your components, it helps you to avoid unexpected bugs.

Validating Props

App.propTypes is used for props validation in react component. When some of the props are passed with an invalid type, you will get the warnings on JavaScript console. After specifying the validation patterns, you will set the App.defaultProps.

Syntax:

```
class App extends React.Component {
        render() {}
}
Component.propTypes = { /*Definition */};
```

ReactJS Props Validator

ReactJS props validator contains the following list of validators.

SN	PropsType	Description
1.	PropTypes.any	The props can be of any data type.

2.	PropTypes.array	The props should be an array.
3.	PropTypes.bool	The props should be a boolean.
4.	PropTypes.func	The props should be a function.
5.	PropTypes.number	The props should be a number.
6.	PropTypes.object	The props should be an object.
7.	PropTypes.string	The props should be a string.
8.	PropTypes.symbol	The props should be a symbol.
9.	PropTypes.instanceOf	The props should be an instance of a particular JavaScript class.
10.	PropTypes.isRequired	The props must be provided.
11.	PropTypes.element	The props must be an element.
12.	PropTypes.node	The props can render anything: numbers, strings, elements or an array (or fragment)

		containing these types.
13.	PropTypes.oneOf()	The props should be one of several types of specific values.
14.	PropTypes.oneOfType([PropTypes.string,PropTypes.number])	The props should be an object that could be one of many types.

Example

Here, we are creating an App component which contains all the props that we need. In this example, **App.propTypes** is used for props validation. For props validation, you must have to add this line: **import PropTypes from 'prop-types'** in **App.js file**.

App.js

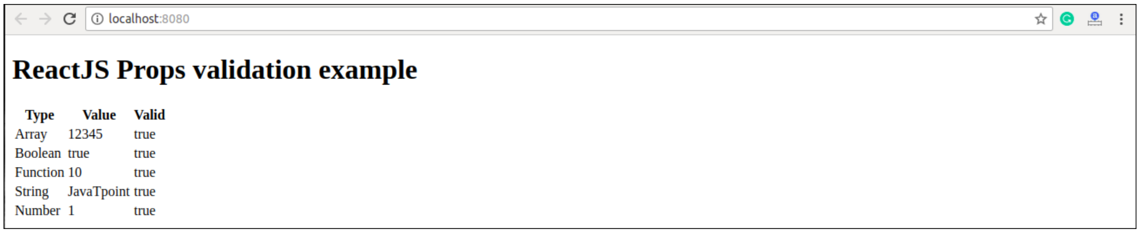
```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>ReactJS Props validation example</h1>
        <table>
          <tr>
            <th>Type</th>
            <th>Value</th>
            <th>Valid</th>
          </tr>
         <tr>
            <td>Array</td>
            <td>{this.props.propArray}</td>
            <td>{this.props.propArray ? "true" : "False"}</td>
          </tr>
```

```
                <tr>
                    <td>Boolean</td>
                    <td>{this.props.propBool ? "true" : "False"}</td>
                    <td>{this.props.propBool ? "true" : "False"}</td>
                </tr>
                <tr>
                    <td>Function</td>
                    <td>{this.props.propFunc(5)}</td>
                    <td>{this.props.propFunc(5) ? "true" : "False"}</td>
                </tr>
                <tr>
                    <td>String</td>
                    <td>{this.props.propString}</td>
                    <td>{this.props.propString ? "true" : "False"}</td>
                </tr>
                <tr>
                    <td>Number</td>
                    <td>{this.props.propNumber}</td>
                    <td>{this.props.propNumber ? "true" : "False"}</td>
                </tr>
            </table>
        </div>
        );
    }
}
App.propTypes = {
    propArray: PropTypes.array.isRequired,
    propBool: PropTypes.bool.isRequired,
    propFunc: PropTypes.func,
    propNumber: PropTypes.number,
    propString: PropTypes.string,
}
App.defaultProps = {
    propArray: [1,2,3,4,5],
    propBool: true,
    propFunc: function(x){return x+5},
    propNumber: 1,
    propString: "JavaTpoint",
}
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'));
```

Output:



ReactJS Props validation example

Type	Value	Valid
Array	12345	true
Boolean	true	true
Function	10	true
String	JavaTpoint	true
Number	1	true

ReactJS Custom Validators

ReactJS allows creating a custom validation function to perform custom validation. The following argument is used to create a custom validation function.

- **props:** It should be the first argument in the component.
- **propName:** It is the propName that is going to validate.
- **componentName:** It is the componentName that are going to validated again.

Example

```
var Component = React.createClass({
App.propTypes = {
  customProp: function(props, propName, componentName) {
      if (!item.isValid(props[propName])) {
        return new Error('Validation failed!');
      }
    }
  }
})
```

Unit-16

State Vs. Props

State

The state is an updatable structure that is used to contain data or information about the component and can change over time. The change in state can happen as a response to user action or system event. It is the heart of the react component which determines the behavior of the component and how it will render. A state must be kept as simple as possible. It represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly.

Props

Props 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 allows passing data from one component to other components. It is similar to function arguments and can be passed to the component the same way as arguments passed in a function. Props are immutable so we cannot modify the props from inside the component.

Difference between State and Props

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.

5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

The below table will guide you about the changing in props and state.

SN	Condition	Props	State
1.	Can get initial value from parent Component?	Yes	Yes
2.	Can be changed by parent Component?	Yes	No
3.	Can set default values inside Component?	Yes	Yes
4.	Can change inside Component?	No	Yes
5.	Can set initial value for child Components?	Yes	Yes
6.	Can change in child Components?	Yes	No

Note: The component State and Props share some common similarities. They are given in the below table.

SN	State and Props

1.	Both are plain JS object.
2.	Both can contain default values.
3.	Both are read-only when they are using by this.

Unit-17

What is Constructor?

The constructor is a method used to initialize an object's state in a class. It automatically called during the creation of an object in a class.

The concept of a constructor is the same in React. The constructor in a React component is called before the component is mounted. When you implement the constructor for a React component, you need to call **super(props)** method before any other statement. If you do not call super(props) method, **this.props** will be undefined in the constructor and can lead to bugs.

Syntax

```
Constructor(props){
    super(props);
}
```

In React, constructors are mainly used for two purposes:

1. It used for initializing the local state of the component by assigning an object to this.state.
2. It used for binding event handler methods that occur in your component.

Note: If you neither initialize state nor bind methods for your React component, there is no need to implement a constructor for React component.

You cannot call **setState()** method directly in the **constructor()**. If the component needs to use local state, you need directly to use '**this.state**' to assign the initial state in the constructor. The constructor only uses this.state to assign initial state, and all other methods need to use set.state() method.

Example

The concept of the constructor can understand from the below example.

App.js

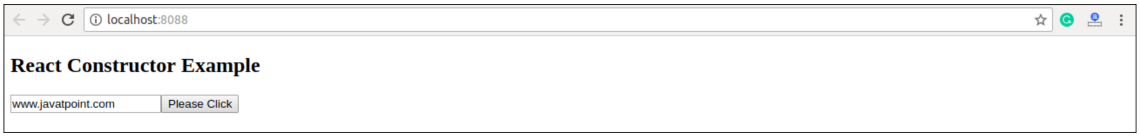
```
import React, { Component } from 'react';

class App extends Component {
  constructor(props){
    super(props);
    this.state = {
         data: 'www.javatpoint.com'
      }
    this.handleEvent = this.handleEvent.bind(this);
  }
  handleEvent(){
    console.log(this.props);
  }
  render() {
    return (
      <div className="App">
    <h2>React Constructor Example</h2>
    <input type ="text" value={this.state.data} />
        <button onClick={this.handleEvent}>Please Click</button>
      </div>
    );
  }
}
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'));
```

Output

When you execute the above code, you get the following output.



React Constructor Example

www.javatpoint.com

The most common question related to the constructor are:

1. Is it necessary to have a constructor in every component?

No, it is not necessary to have a constructor in every component. If the component is not complex, it simply returns a node.

```
class App extends Component {
   render () {
      return (
         <p> Name: { this.props.name }</p>
      );
   }
}
```

2. Is it necessary to call super() inside a constructor?

Yes, it is necessary to call super() inside a constructor. If you need to set a property or access 'this' inside the constructor in your component, you need to call super().

```
class App extends Component {
   constructor(props){
      this.fName = "Jhon"; // 'this' is not allowed before super()
   }
   render () {
      return (
         <p> Name: { this.props.name }</p>
      );
   }
}
```

When you run the above code, you get an error saying **'this' is not allowed before super()**. So if you need to access the props inside the constructor, you need to call super(props).

Arrow Functions

The Arrow function is the new feature of the ES6 standard. If you need to use arrow functions, it is not necessary to bind any event to 'this.' Here, the scope of 'this' is global and not limited to any calling function. So If you are using Arrow Function, there is no need to bind 'this' inside the constructor.

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

class App extends Component {
  constructor(props){
    super(props);
    this.state = {
        data: 'www.javatpoint.com'
      }
  }
  handleEvent = () => {
    console.log(this.props);
  }
  render() {
    return (
      <div className="App">
    <h2>React Constructor Example</h2>
    <input type ="text" value={this.state.data} />
        <button onClick={this.handleEvent}>Please Click</button>
      </div>
    );
  }
}
export default App;
```

We can use a constructor in the following ways:

1) The constructor is used to initialize state.

```
class App extends Component {
  constructor(props){
        // here, it is setting initial value for 'inputTextValue'
        this.state = {
            inputTextValue: 'initial value',
        };
  }
}
```

2) Using 'this' inside constructor

```
class App extends Component {
    constructor(props) {
        // when you use 'this' in constructor, super() needs to be called first
        super();
        // it means, when you want to use 'this.props' in constructor, call it as below
        super(props);
    }
}
```

3) Initializing third-party libraries

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

        this.myBook = new MyBookLibrary();

        //Here, you can access props without using 'this'
        this.Book2 = new MyBookLibrary(props.environment);
    }
}
```

4) Binding some context(this) when you need a class method to be passed in props to children.

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

        // when you need to 'bind' context to a function
        this.handleFunction = this.handleFunction.bind(this);
    }
}
```

Unit-18

React Component API

ReactJS component is a top-level API. It makes the code completely individual and reusable in the application. It includes various methods for:

- Creating elements
- Transforming elements
- Fragments

Here, we are going to explain the three most important methods available in the React component API.

1. setState()
2. forceUpdate()
3. findDOMNode()

setState()

This method is used to update the state of the component. This method does not always replace the state immediately. Instead, it only adds changes to the original state. It is a primary method that is used to update the user interface(UI) in response to event handlers and server responses.

> ***Note: In the ES6 classes, this.method.bind(this) is used to manually bind the setState() method.***

Syntax

```
this.stateState(object newState[, function callback]);
```

In the above syntax, there is an optional **callback** function which is executed once setState() is completed and the component is re-rendered.

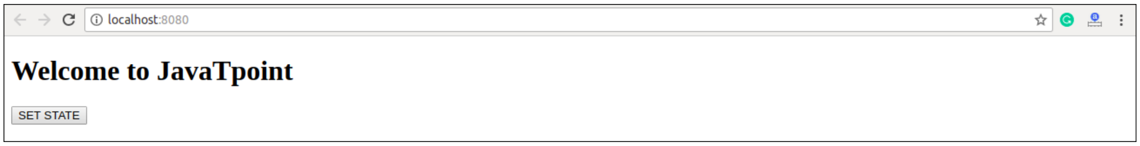
Example

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
class App extends React.Component {
   constructor() {
      super();
      this.state = {
         msg: "Welcome to JavaTpoint"
      };
      this.updateSetState = this.updateSetState.bind(this);
   }
   updateSetState() {
       this.setState({
         msg:"Its a best ReactJS tutorial"
       });
   }
   render() {
      return (
         <div>
            <h1>{this.state.msg}</h1>
            <button onClick = {this.updateSetState}>SET STATE</button>
         </div>
      );
   }
}
export default App;
```

Main.js

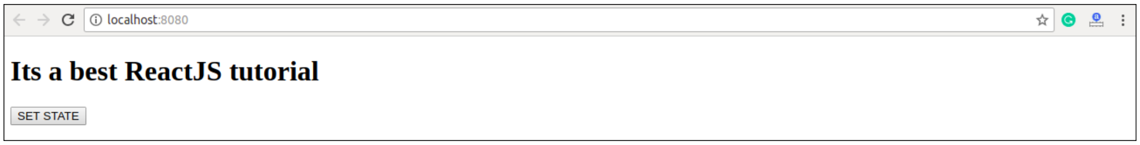
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

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

Output:



Welcome to JavaTpoint

When you click on the **SET STATE** button, you will see the following screen with the updated message.



Its a best ReactJS tutorial

forceUpdate()

This method allows us to update the component manually.

Syntax

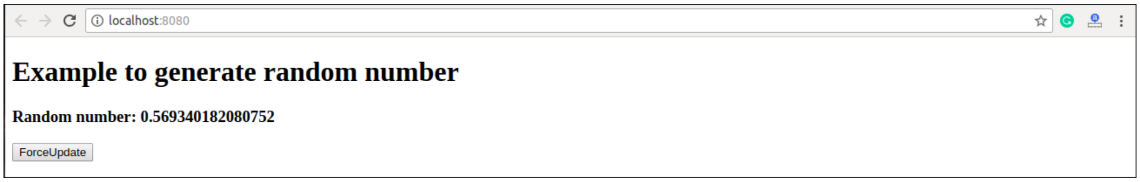
Component.forceUpdate(callback);

Example

App.js

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor() {
    super();
    this.forceUpdateState = this.forceUpdateState.bind(this);
  }
  forceUpdateState() {
    this.forceUpdate();
  };
  render() {
    return (
      <div>
        <h1>Example to generate random number</h1>
        <h3>Random number: {Math.random()}</h3>
        <button onClick = {this.forceUpdateState}>ForceUpdate</button>
      </div>
    );
  }
}
export default App;
```

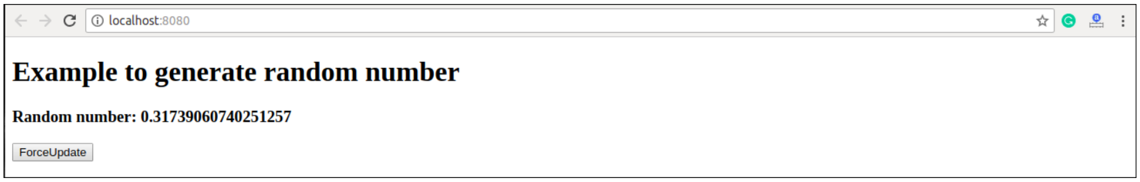
Output:



Example to generate random number

Random number: 0.569340182080752

Each time when you click on **ForceUpdate** button, it will generate the **random** number. It can be shown in the below image.



Example to generate random number

Random number: 0.31739060740251257

findDOMNode()

For DOM manipulation, you need to use **ReactDOM.findDOMNode()** method. This method allows us to find or access the underlying DOM node.

Syntax

```
ReactDOM.findDOMNode(component);
```

Example

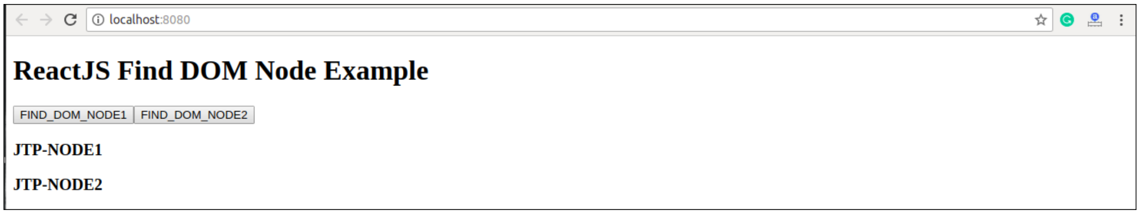
For DOM manipulation, first, you need to import this line: **import ReactDOM** from '**react-dom**' in your **App.js** file.

App.js

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component {
  constructor() {
    super();
    this.findDomNodeHandler1 = this.findDomNodeHandler1.bind(this);
    this.findDomNodeHandler2 = this.findDomNodeHandler2.bind(this);
  };
  findDomNodeHandler1() {
    var myDiv = document.getElementById('myDivOne');
    ReactDOM.findDOMNode(myDivOne).style.color = 'red';
  }
  findDomNodeHandler2() {
    var myDiv = document.getElementById('myDivTwo');
    ReactDOM.findDOMNode(myDivTwo).style.color = 'blue';
  }
  render() {
    return (
      <div>
        <h1>ReactJS Find DOM Node Example</h1>
        <button onClick = {this.findDomNodeHandler1}>FIND_DOM_NODE1</button>

        <button onClick = {this.findDomNodeHandler2}>FIND_DOM_NODE2</button>
```

```
            <h3 id = "myDivOne">JTP-NODE1</h3>
            <h3 id = "myDivTwo">JTP-NODE2</h3>
         </div>
      );
   }
}
export default App;
```

Output:

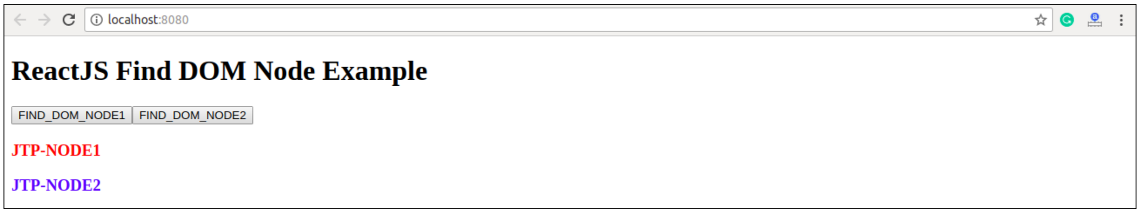


ReactJS Find DOM Node Example

JTP-NODE1

JTP-NODE2

Once you click on the **button**, the color of the node gets changed. It can be shown in the below screen.



ReactJS Find DOM Node Example

JTP-NODE1

JTP-NODE2

Unit-19

React Component Life-Cycle

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and called at various points during a component's life. The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase

Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.

1. Initial Phase

It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

- **getDefaultProps()**
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.
- **getInitialState()**
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

2. Mounting Phase

In this phase, the instance of a component is created and inserted into the DOM. It consists of the following methods.

- **componentWillMount()**
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
- **componentDidMount()**
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
- **render()**
  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

3. Updating Phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

- **componentWillRecieveProps()**
  It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.
- **shouldComponentUpdate()**
  It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.
- **componentWillUpdate()**
  It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method. It will not be called, if **shouldComponentUpdate()** returns false.
- **render()**
  It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number. If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.
- **componentDidUpdate()**
  It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

4. Unmounting Phase

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.

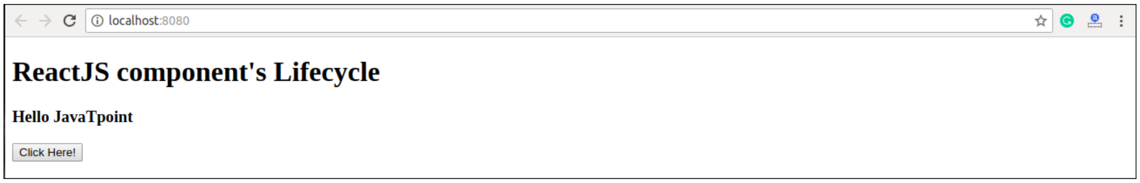
- **componentWillUnmount()**

  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

Example

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

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {hello: "JavaTpoint"};
    this.changeState = this.changeState.bind(this)
  }
  render() {
    return (
      <div>
        <h1>ReactJS component's Lifecycle</h1>
        <h3>Hello {this.state.hello}</h3>
        <button onClick = {this.changeState}>Click Here!</button>
      </div>
    );
  }
  componentWillMount() {
    console.log('Component Will MOUNT!')
  }
  componentDidMount() {
    console.log('Component Did MOUNT!')
  }
  changeState(){
    this.setState({hello:"All!!- Its a great reactjs tutorial."});
  }
  componentWillReceiveProps(newProps) {
    console.log('Component Will Recieve Props!')
  }
  shouldComponentUpdate(newProps, newState) {
    return true;
  }
  componentWillUpdate(nextProps, nextState) {
    console.log('Component Will UPDATE!');
  }
  componentDidUpdate(prevProps, prevState) {
    console.log('Component Did UPDATE!')
  }
  componentWillUnmount() {
    console.log('Component Will UNMOUNT!')
  }
}
export default App;
```

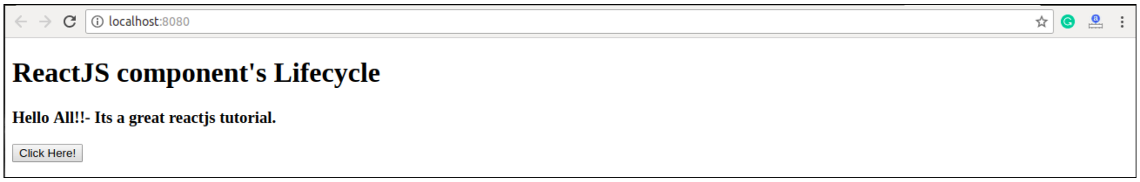
Output:



ReactJS component's Lifecycle

Hello JavaTpoint

When you click on the **Click Here** Button, you get the updated result which is shown in the below screen.



ReactJS component's Lifecycle

Hello All!!- Its a great reactjs tutorial.

Unit-20

React Forms

Forms are an integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users. Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

Creating Form

React offers a stateful, reactive approach to build a form. The component rather than the DOM usually handles the React form. In React, the form is usually implemented by using controlled components.

There are mainly two types of form input in React.

1. Uncontrolled component
2. Controlled component

Uncontrolled component

The uncontrolled input is similar to the traditional HTML form inputs. The DOM itself handles the form data. Here, the HTML elements maintain their own state that will be updated when the input value changes. To write an uncontrolled component, you need to use a ref to get form values from the DOM. In other words, there is no need to write an event handler for every state update. You can use a ref to access the input field value of the form from the DOM.

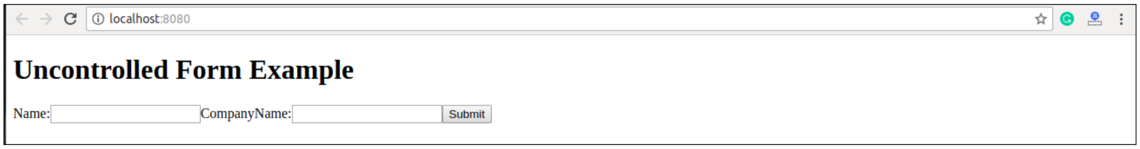
Example

In this example, the code accepts a field **username** and **company name** in an uncontrolled component.

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
      super(props);
      this.updateSubmit = this.updateSubmit.bind(this);
      this.input = React.createRef();
  }
  updateSubmit(event) {
      alert('You have entered the UserName and CompanyName successfully.');
      event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.updateSubmit}>
        <h1>Uncontrolled Form Example</h1>
        <label>Name:
            <input type="text" ref={this.input} />
        </label>
        <label>
            CompanyName:
            <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
export default App;
```

Output

When you execute the above code, you will see the following screen.

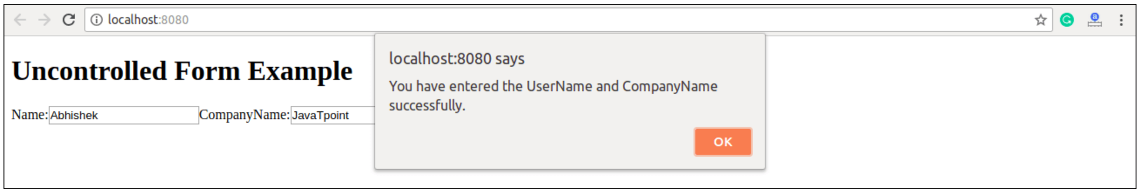


localhost:8080

Uncontrolled Form Example

Name: CompanyName: Submit

After filling the data in the field, you get the message that can be seen in the below screen.



localhost:8080

Uncontrolled Form Example

Name: Abhishek CompanyName: JavaTpoint

localhost:8080 says

You have entered the UserName and CompanyName successfully.

OK

Controlled Component

In HTML, form elements typically maintain their own state and update it according to the user input. In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with **setState()** method.

Controlled components have functions that govern the data passing into them on every **onChange event**, rather than grabbing the data only once, e.g., when you click a **submit button**. This data is then saved to state and updated with setState() method. This makes component have better control over the form elements and data.

A controlled component takes its current value through **props** and notifies the changes through **callbacks** like an onChange event. A parent component "controls" this changes by handling the callback and managing its own state and then passing the new values as props to the controlled component. It is also called as a "dumb component."

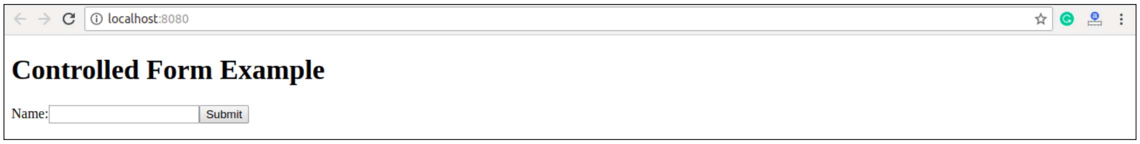
Example

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
      super(props);
      this.state = {value: ''};
      this.handleChange = this.handleChange.bind(this);
      this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
      this.setState({value: event.target.value});
  }
  handleSubmit(event) {
      alert('You have submitted the input successfully: ' + this.state.value);
      event.preventDefault();
  }
  render() {
      return (
          <form onSubmit={this.handleSubmit}>
            <h1>Controlled Form Example</h1>
            <label>
```

```
        Name:
     <input type="text" value={this.state.value} onChange={this.handleChange} />
      </label>
      <input type="submit" value="Submit" />
     </form>
   );
 }
}
export default App;
```

Output

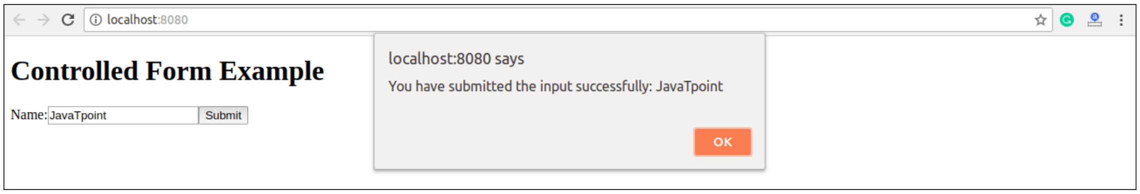
When you execute the above code, you will see the following screen.



Controlled Form Example

Name:

After filling the data in the field, you get the message that can be seen in the below screen.



Controlled Form Example

Name: JavaTpoint

localhost:8080 says

You have submitted the input successfully: JavaTpoint

Handling Multiple Inputs in Controlled Component

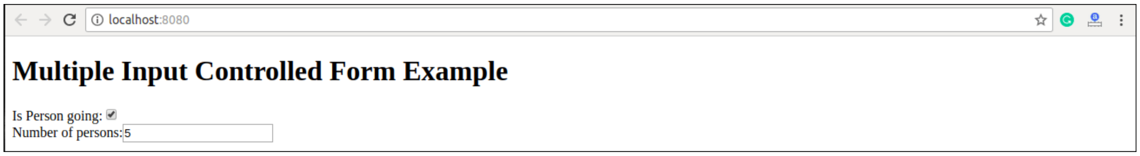
If you want to handle multiple controlled input elements, add a **name** attribute to each element, and then the handler function decided what to do based on the value of **event.target.name**.

Example

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      personGoing: true,
      numberOfPersons: 5
    };
    this.handleInputChange = this.handleInputChange.bind(this);
```

```
   }
   handleInputChange(event) {
       const target = event.target;
       const value = target.type === 'checkbox' ? target.checked : target.value;
       const name = target.name;
       this.setState({
           [name]: value
       });
  }
  render() {
      return (
          <form>
              <h1>Multiple Input Controlled Form Example</h1>
              <label>
                  Is Person going:
                  <input
                    name="personGoing"
                    type="checkbox"
                    checked={this.state.personGoing}
                    onChange={this.handleInputChange} />
             </label>
             <br />
             <label>
                 Number of persons:
                 <input
                 name="numberOfPersons"
                 type="number"
                 value={this.state.numberOfPersons}
                 onChange={this.handleInputChange} />
             </label>
         </form>
     );
  }
}
export default App;
```

Output



localhost:8080

Multiple Input Controlled Form Example

Is Person going: ☑

Number of persons: 5

Unit-21

React Controlled Vs. Uncontrolled Component

Controlled Component

A controlled component is bound to a value, and its changes will be handled in code by using **event-based callbacks**. Here, the input form element is handled by the react itself rather than the DOM. In this, the mutable state is kept in the state property and will be updated only with setState() method.

Controlled components have functions that govern the data passing into them on every **onChange** event occurs. This data is then saved to state and updated with setState() method. It makes component have better control over the form elements and data.

Uncontrolled Component

It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself. It maintains their own state and will be updated when the input value changes. To write an uncontrolled component, there is no need to write an event handler for every state update, and you can use a ref to access the value of the form from the DOM.

Difference table between controlled and uncontrolled component

SN	Controlled	Uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.

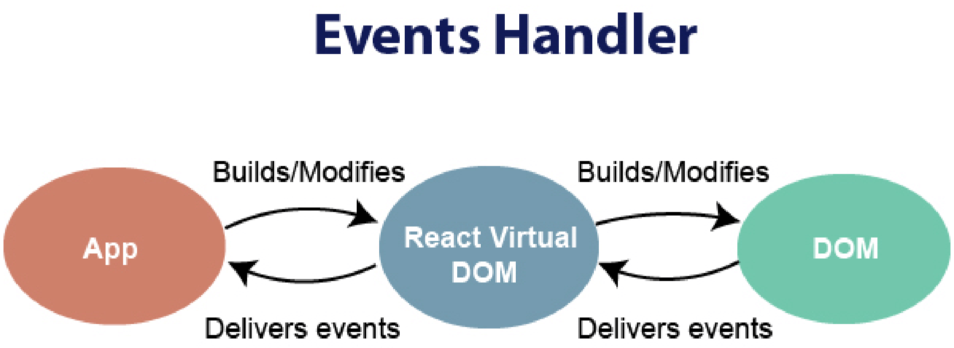
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

Unit-22

React Events

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.



Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**. For example:

Event declaration in plain HTML:

```
<button onclick="showMessage()">
       Hello JavaTpoint
</button>
```

Event declaration in React:

```
<button onClick={showMessage}>
      Hello JavaTpoint
</button>
```

In react, we cannot return **false** to prevent the **default** behavior. We must call **preventDefault** event explicitly to prevent the default behavior. For example:

In plain HTML, to prevent the default link behavior of opening a new page, we can write:

```
<a href="#" onclick="console.log('You had clicked a Link.'); return false">
    Click_Me
</a>
```

In React, we can write it as:

```
function ActionLink() {
    function handleClick(e) {
        e.preventDefault();
        console.log('You had clicked a Link.');
    }
    return (
        <a href="#" onClick={handleClick}>
              Click_Me
        </a>
    );
}
```

In the above example, e is a **Synthetic Event** which defines according to the **W3C** spec.

Now let us see how to use Event in React.

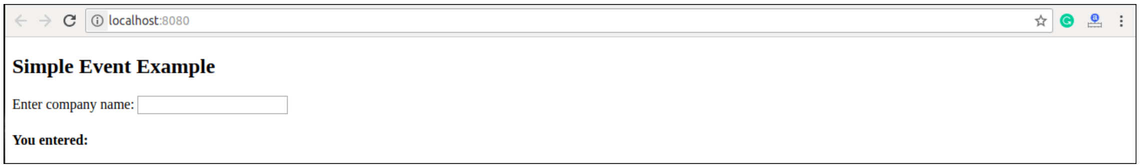
Example

In the below example, we have used only one component and adding an onChange event. This event will trigger the **changeText** function, which returns the company name.

```
import React, { Component } from 'react';
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            companyName: ''
        };
    }
    changeText(event) {
        this.setState({
            companyName: event.target.value
        });
    }
    render() {
        return (
            <div>
                <h2>Simple Event Example</h2>
                <label htmlFor="name">Enter company name: </label>
                <input type="text" id="companyName" onChange={this.changeText.bind(this)}/>
                <h4>You entered: { this.state.companyName }</h4>
            </div>
        );
    }
}
export default App;
```

Output

When you execute the above code, you will get the following output.

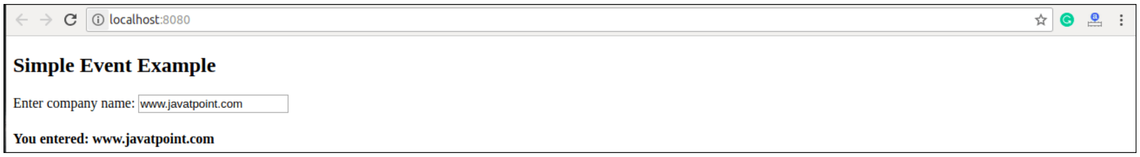


Simple Event Example

Enter company name:

You entered:

After entering the name in the textbox, you will get the output as like below screen.



Simple Event Example

Enter company name: www.javatpoint.com

You entered: www.javatpoint.com

Unit-23

React Conditional Rendering

In React, we can create multiple components which encapsulate behaviour that we need. After that, we can render them depending on some conditions or the state of our application. In other words, based on one or several conditions, a component decides which elements it will return. In React, conditional rendering works the same way as the conditions work in JavaScript. We use JavaScript operators to create elements representing the current state, and then React Component update the UI to match them.

From the given scenario, we can understand how conditional rendering works. Consider an example of handling a **login/logout** button. The login and logout buttons will be separate components. If a user logged in, render the **logout component** to display the logout button. If a user not logged in, render the **login component** to display the login button. In React, this situation is called as **conditional rendering**.

There is more than one way to do conditional rendering in React. They are given below.

- if
- ternary operator
- logical && operator
- switch case operator
- Conditional Rendering with enums

if

It is the easiest way to have a conditional rendering in React in the render method. It is restricted to the total block of the component. IF the condition is **true**, it will return the element to be rendered. It can be understood in the below example.

Example

```
function UserLoggin(props) {
return <h1>Welcome back!</h1>;
}
function GuestLoggin(props) {
  return <h1>Please sign up.</h1>;
}
function SignUp(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserLogin />;
  }
  return <GuestLogin />;
}

ReactDOM.render(
  <SignUp isLoggedIn={false} />,
  document.getElementById('root')
);
```

Logical && operator

This operator is used for checking the condition. If the condition is **true**, it will return the element **right** after **&&**, and if it is **false**, React will **ignore** and skip it.

Syntax

```
{
   condition &&
   // whatever written after && will be a part of output.
}
```

We can understand the behavior of this concept from the below example.

If you run the below code, you will not see the **alert** message because the condition is not matching.

```
('javatpoint' == 'JavaTpoint') && alert('This alert will never be shown!')
```

If you run the below code, you will see the **alert** message because the condition is matching.

```
(10 > 5) && alert('This alert will be shown!')
```

Example

```
import React from 'react';
import ReactDOM from 'react-dom';
// Example Component
function Example()
{
   return(<div>
         {
            (10 > 5) && alert('This alert will be shown!')
         }
        </div>
     );
}
```

You can see in the above output that as the condition **(10 > 5)** evaluates to true, the alert message is successfully rendered on the screen.

Ternary operator

The ternary operator is used in cases where two blocks alternate given a certain condition. This operator makes your if-else statement more concise. It takes **three** operands and used as a shortcut for the if statement.

Syntax

```
condition ?  true : false
```

If the condition is **true**, **statement1** will be rendered. Otherwise, **false** will be rendered.

Example

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      Welcome {isLoggedIn ? 'Back' : 'Please login first'}.
    </div>
  );
}
```

Switch case operator

Sometimes it is possible to have multiple conditional renderings. In the switch case, conditional rendering is applied based on a different state.

Example

```
function NotificationMsg({ text}) {
  switch(text) {
    case 'Hi All':
      return <Message: text={text} />;
    case 'Hello JavaTpoint':
      return <Message text={text} />;
    default:
      return null;
  }
}
```

Conditional Rendering with enums

An **enum** is a great way to have a multiple conditional rendering. It is more **readable** as compared to switch case operator. It is perfect for **mapping** between different **state**. It is also perfect for mapping in more than one condition. It can be understood in the below example.

Example

```
function NotificationMsg({ text, state }) {
  return (
    <div>
      {{
        info: <Message text={text} />,
        warning: <Message text={text} />,
      }[state]}
    </div>
  );
}
```

Conditional Rendering Example

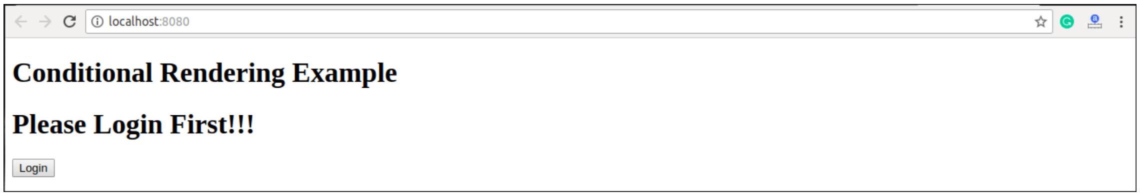
In the below example, we have created a **stateful** component called **App** which maintains the login control. Here, we create three components representing Logout, Login, and Message component. The stateful component App will render either or depending on its current **state**.

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

```
// Message Component
function Message(props)
{
   if (props.isLoggedIn)
      return <h1>Welcome Back!!!</h1>;
   else
      return <h1>Please Login First!!!</h1>;
}
// Login Component
function Login(props)
{
  return(
        <button onClick = {props.clickInfo}> Login </button>
     );
}
// Logout Component
function Logout(props)
{
   return(
        <button onClick = {props.clickInfo}> Logout </button>
     );
}
class App extends Component{
  constructor(props)
   {
      super(props);
      this.handleLogin = this.handleLogin.bind(this);
        this.handleLogout = this.handleLogout.bind(this);
    this.state = {isLoggedIn : false};
   }
  handleLogin()
   {
      this.setState({isLoggedIn : true});
   }
   handleLogout()
   {
      this.setState({isLoggedIn : false});
   }
   render(){
      return(
         <div>
      <h1> Conditional Rendering Example </h1>
            <Message isLoggedIn = {this.state.isLoggedIn}/>
            {
                    (this.state.isLoggedIn)?(
                    <Logout clickInfo = {this.handleLogout} />
                    ) : (
                    <Login clickInfo = {this.handleLogin} />
                    )
                }
            </div>
            );
    }
}
export default App;
```

Output:

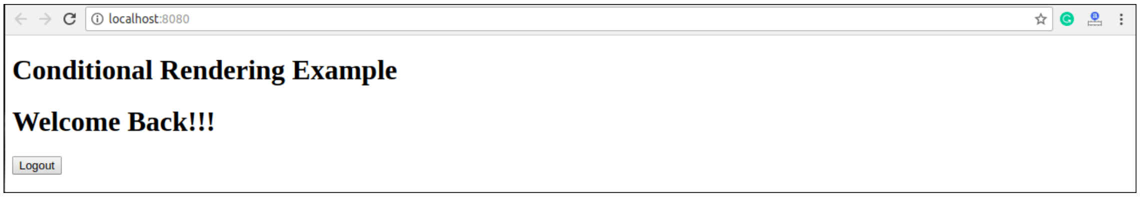
When you execute the above code, you will get the following screen.



Conditional Rendering Example

Please Login First!!!

After clicking the logout button, you will get the below screen.



Conditional Rendering Example

Welcome Back!!!

Preventing Component form Rendering

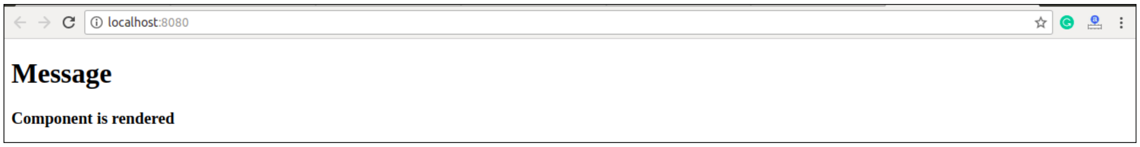
Sometimes it might happen that a component hides itself even though another component rendered it. To do this (prevent a component from rendering), we will have to return **null** instead of its render output. It can be understood in the below example:

Example

In this example, the is rendered based on the value of the prop called **displayMessage**. If the prop value is false, then the component does not render.

```
import React from 'react';
import ReactDOM from 'react-dom';
function Show(props)
{
    if(!props.displayMessage)
        return null;
    else
        return <h3>Component is rendered</h3>;
}
ReactDOM.render(
    <div>
        <h1>Message</h1>
        <Show displayMessage = {true} />
    </div>,
    document.getElementById('app')
);
```

Output:



Message

Component is rendered

Unit-24

React Lists

Lists are used to display data in an ordered format and mainly used to display menus on websites. In React, Lists can be created in a similar way as we create lists in JavaScript. Let us see how we transform Lists in regular JavaScript.

The map() function is used for traversing the lists. In the below example, the map() function takes an array of numbers and multiply their values with 5. We assign the new array returned by map() to the variable multiplyNums and log it.

Example

```
var numbers = [1, 2, 3, 4, 5];
const multiplyNums = numbers.map((number)=>{
    return (number * 5);
});
console.log(multiplyNums);
```

Output

The above JavaScript code will log the output on the console. The output of the code is given below.

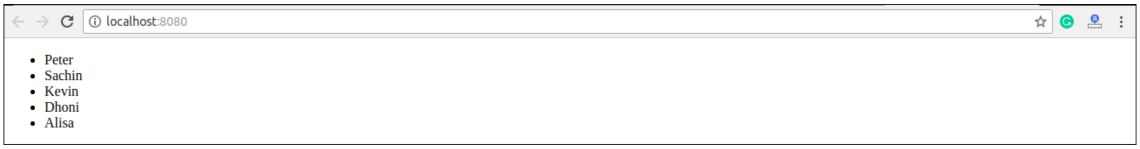
```
[5, 10, 15, 20, 25]
```

Now, let us see how we create a list in React. To do this, we will use the map() function for traversing the list element, and for updates, we enclose them between **curly braces { }**. Finally, we assign the array elements to listItems. Now, include this new list inside **<ul> </ul>** elements and render it to the DOM.

Example

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

const myList = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
const listItems = myList.map((myList)=>{
   return <li>{myList}</li>;
});
ReactDOM.render(
   <ul> {listItems} </ul>,
   document.getElementById('app')
);
export default App;
```

Output



- Peter
- Sachin
- Kevin
- Dhoni
- Alisa

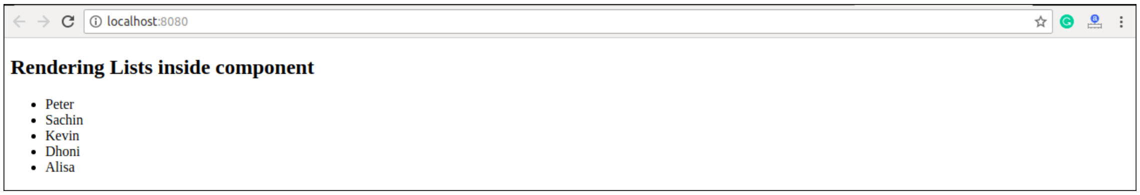
Rendering Lists inside components

In the previous example, we had directly rendered the list to the DOM. But it is not a good practice to render lists in React. In React, we had already seen that everything is built as individual components. Hence, we would need to render lists inside a component. We can understand it in the following code.

Example

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

function NameList(props) {
  const myLists = props.myLists;
  const listItems = myLists.map((myList) =>
    <li>{myList}</li>
  );
  return (
    <div>
        <h2>Rendering Lists inside component</h2>
              <ul>{listItems}</ul>
    </div>
  );
}
const myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
ReactDOM.render(
  <NameList myLists={myLists} />,
  document.getElementById('app')
);
export default App;
```

Output



Rendering Lists inside component

- Peter
- Sachin
- Kevin
- Dhoni
- Alisa

Unit-25

React Keys

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time.

Keys should be given inside the array to give the elements a stable identity. The best way to pick a key as a string that uniquely identifies the items in the list. It can be understood with the below example.

Example

```
const stringLists = [ 'Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa' ];

const updatedLists = stringLists.map((strList)=>{
    <li key={strList.id}> {strList} </li>;
});
```

If there are no stable IDs for rendered items, you can assign the item **index** as a key to the lists. It can be shown in the below example.

Example

```
const stringLists = [ 'Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa' ];

const updatedLists = stringLists.map((strList, index)=>{
    <li key={index}> {strList} </li>;
});
```

Note: It is not recommended to use indexes for keys if the order of the item may change in future. It creates confusion for the developer and may cause issues with the component state.

Using Keys with component

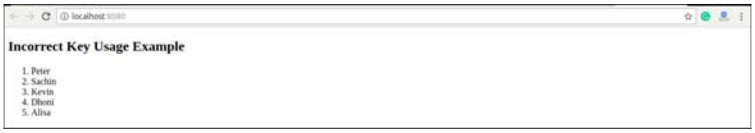
Consider you have created a separate component for **ListItem** and extracting ListItem from that component. In this case, you should have to assign keys on the **<ListItem />** elements in the array, not to the **<li>** elements in the ListItem itself. To avoid mistakes, you have to keep in mind that keys only make sense in the context of the surrounding array. So, anything you are returning from map() function is recommended to be assigned a key.

Example: Incorrect Key usage

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

function ListItem(props) {
  const item = props.item;
  return (
    // Wrong! No need to specify the key here.
    <li key={item.toString()}>
      {item}
    </li>
  );
}
function NameList(props) {
  const myLists = props.myLists;
  const listItems = myLists.map((strLists) =>
    // The key should have been specified here.
    <ListItem item={strLists} />
  );
  return (
    <div>
        <h2>Incorrect Key Usage Example</h2>
              <ol>{listItems}</ol>
    </div>
  );
}
const myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
ReactDOM.render(
  <NameList myLists={myLists}/>,
  document.getElementById('app')
);
export default App;
```

In the given example, the list is rendered successfully. But it is not a good practice that we had not assigned a key to the map() iterator.

Output



Incorrect Key Usage Example

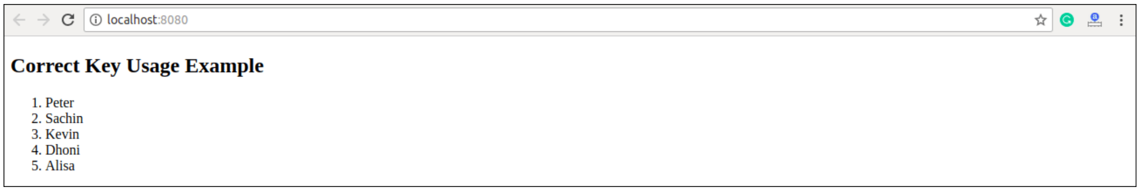
1. Peter
2. Sachin
3. Kevin
4. Dhoni
5. Alisa

Example: Correct Key usage

To correct the above example, we should have to assign key to the map() iterator.

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

function ListItem(props) {
  const item = props.item;
  return (
    // No need to specify the key here.
    <li> {item} </li>
  );
}
function NameList(props) {
  const myLists = props.myLists;
  const listItems = myLists.map((strLists) =>
    // The key should have been specified here.
    <ListItem key={myLists.toString()} item={strLists} />
  );
  return (
    <div>
        <h2>Correct Key Usage Example</h2>
            <ol>{listItems}</ol>
    </div>
  );
}
const myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
ReactDOM.render(
  <NameList myLists={myLists}/>,
  document.getElementById('app')
);
export default App;
```

Output



Correct Key Usage Example

1. Peter
2. Sachin
3. Kevin
4. Dhoni
5. Alisa

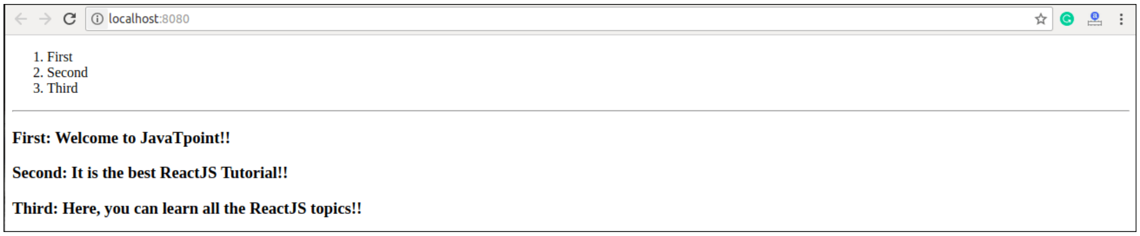
Uniqueness of Keys among Siblings

We had discussed that keys assignment in arrays must be unique among their **siblings**. However, it doesn't mean that the keys should be **globally** unique. We can use the same set of keys in producing two different arrays. It can be understood in the below example.

Example

```
import React from 'react';
import ReactDOM from 'react-dom';
function MenuBlog(props) {
  const titlebar = (
    <ol>
      {props.data.map((show) =>
        <li key={show.id}>
          {show.title}
        </li>
      )}
    </ol>
  );
  const content = props.data.map((show) =>
    <div key={show.id}>
      <h3>{show.title}: {show.content}</h3>
    </div>
  );
  return (
    <div>
      {titlebar}
      <hr />
      {content}
    </div>
  );
}
const data = [
  {id: 1, title: 'First', content: 'Welcome to JavaTpoint!!'},
  {id: 2, title: 'Second', content: 'It is the best ReactJS Tutorial!!'},
  {id: 3, title: 'Third', content: 'Here, you can learn all the ReactJS topics!!'}
];
ReactDOM.render(
  <MenuBlog data={data} />,
  document.getElementById('app')
);
export default App;
```

Output



```
   1. First
   2. Second
   3. Third

First: Welcome to JavaTpoint!!

Second: It is the best ReactJS Tutorial!!

Third: Here, you can learn all the ReactJS topics!!
```

Unit-26

React Refs

Refs is the shorthand used for **references** in React. It is similar to **keys** in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

When to Use Refs

Refs can be used in the following cases:

- When we need DOM measurements such as managing focus, text selection, or media playback.
- It is used in triggering imperative animations.
- When integrating with third-party DOM libraries.
- It can also use as in callbacks.

When to not use Refs

- Its use should be avoided for anything that can be done **declaratively**. For example, instead of using **open()** and **close()** methods on a Dialog component, you need to pass an **isOpen** prop to it.
- You should have to avoid overuse of the Refs.

How to create Refs

In React, Refs can be created by using **React.createRef()**. It can be assigned to React elements via the **ref** attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.callRef = React.createRef();
  }
  render() {
    return <div ref={this.callRef} />;
  }
}
```

How to access Refs

In React, when a ref is passed to an element inside render method, a reference to the node can be accessed via the current attribute of the ref.

```
const node = this.callRef.current;
```

Refs current Properties

The ref value differs depending on the type of the node:

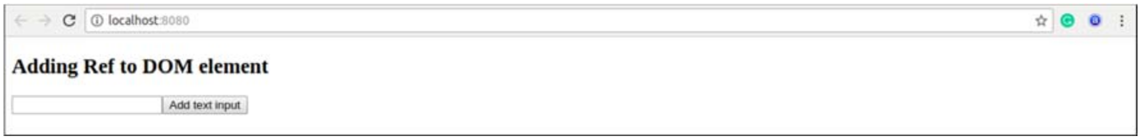
- When the ref attribute is used in HTML element, the ref created with **React.createRef()** receives the underlying DOM element as its **current** property.
- If the ref attribute is used on a custom class component, then ref object receives the **mounted** instance of the component as its current property.
- The ref attribute cannot be used on **function components** because they don't have instances.

Add Ref to DOM elements

In the below example, we are adding a ref to store the reference to a DOM node or element.

```
import React, { Component } from 'react';
import { render } from 'react-dom';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.callRef = React.createRef();
    this.addingRefInput = this.addingRefInput.bind(this);
  }

  addingRefInput() {
    this.callRef.current.focus();
  }

  render() {
    return (
      <div>
        <h2>Adding Ref to DOM element</h2>
        <input
          type="text"
          ref={this.callRef} />
        <input
          type="button"
          value="Add text input"
          onClick={this.addingRefInput}
        />
      </div>
    );
  }
}
export default App;
```

Output



Adding Ref to DOM element

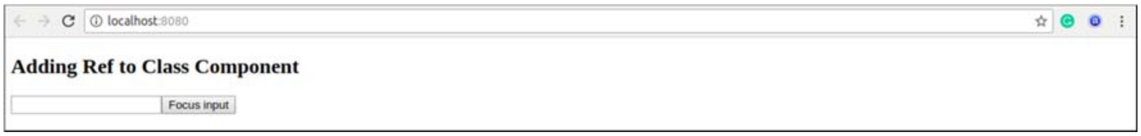
Add Ref to Class components

In the below example, we are adding a ref to store the reference to a class component.

Example

```
import React, { Component } from 'react';
import { render } from 'react-dom';

function CustomInput(props) {
  let callRefInput = React.createRef();

  function handleClick() {
    callRefInput.current.focus();
  }

  return (
    <div>
      <h2>Adding Ref to Class Component</h2>
      <input
        type="text"
        ref={callRefInput} />
      <input
        type="button"
        value="Focus input"
        onClick={handleClick}
      />
    </div>
  );
}
class App extends React.Component {
  constructor(props) {
    super(props);
    this.callRefInput = React.createRef();
  }

  focusRefInput() {
    this.callRefInput.current.focus();
  }

  render() {
    return (
      <CustomInput ref={this.callRefInput} />
    );
  }
}
export default App;
```

Output



Adding Ref to Class Component

Callback refs

In react, there is another way to use refs that is called "**callback refs**" and it gives more control when the refs are **set** and **unset**. Instead of creating refs by createRef() method, React allows a way to create refs by passing a callback function to the ref attribute of a component. It looks like the below code.

```
<input type="text" ref={element => this.callRefInput = element} />
```

The callback function is used to store a reference to the DOM node in an instance property and can be accessed elsewhere. It can be accessed as below:

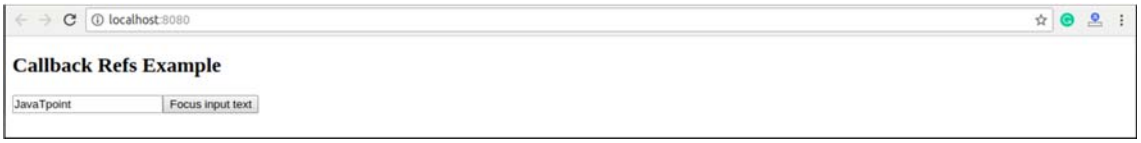
```
this.callRefInput.value
```

The example below helps to understand the working of callback refs.

```
import React, { Component } from 'react';
import { render } from 'react-dom';

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

    this.callRefInput = null;

    this.setInputRef = element => {
      this.callRefInput = element;
    };

    this.focusRefInput = () => {
      //Focus the input using the raw DOM API
      if (this.callRefInput) this.callRefInput.focus();
    };
  }

  componentDidMount() {
    //autofocus of the input on mount
    this.focusRefInput();
  }

  render() {
    return (
      <div>
    <h2>Callback Refs Example</h2>
        <input
          type="text"
          ref={this.setInputRef}
        />
        <input
          type="button"
          value="Focus input text"
          onClick={this.focusRefInput}
        />
      </div>
    );
  }
}
export default App;
```

In the above example, React will call the "ref" callback to store the reference to the input DOM element when the component **mounts**, and when the component **unmounts**, call it with **null**. Refs are always **up-to-date** before the **componentDidMount** or **componentDidUpdate** fires. The callback refs pass between components is the same as you can work with object refs, which is created with React.createRef().

Output



Callback Refs Example

JavaTpoint

Forwarding Ref from one component to another component

Ref forwarding is a technique that is used for passing a **ref** through a component to one of its child components. It can be performed by making use of the **React.forwardRef()** method. This technique is particularly useful with **higher-order components** and specially used in reusable component libraries. The most common example is given below.

Example

```
import React, { Component } from 'react';
import { render } from 'react-dom';

const TextInput = React.forwardRef((props, ref) => (
  <input type="text" placeholder="Hello World" ref={ref} />
));

const inputRef = React.createRef();

class CustomTextInput extends React.Component {
  handleSubmit = e => {
    e.preventDefault();
    console.log(inputRef.current.value);
  };
  render() {
    return (
      <div>
        <form onSubmit={e => this.handleSubmit(e)}>
          <TextInput ref={inputRef} />
          <button>Submit</button>
        </form>
      </div>
    );
  }
}
export default App;
```

In the above example, there is a component **TextInput** that has a child as an input field. Now, to pass or forward the **ref** down to the input, first, create a ref and then pass your ref down to **<TextInput ref={inputRef}>**. After that, React forwards the ref to the **forwardRef** function as a second argument. Next, we forward this ref argument down to **<input ref={ref}>**. Now, the value of the DOM node can be accessed at **inputRef.current**.

React with useRef()

It is introduced in **React 16.7** and above version. It helps to get access the DOM node or element, and then we can interact with that DOM node or element such as focussing the input element or accessing the input element value. It returns the ref object whose **.current** property initialized to the passed argument. The returned object persist for the lifetime of the component.

Syntax

```
const refContainer = useRef(initialValue);
```

Example

In the below code, **useRef** is a function that gets assigned to a variable, **inputRef**, and then attached to an attribute called ref inside the HTML element in which you want to reference.

```
function useRefExample() {
  const inputRef= useRef(null);
  const onButtonClick = () => {
    inputRef.current.focus();
  };
  return (
    <>
      <input ref={inputRef} type="text" />
      <button onClick={onButtonClick}>Submit</button>
    </>
  );
}
```

Unit-27

React Fragments

In React, whenever you want to render something on the screen, you need to use a render method inside the component. This render method can return **single** elements or **multiple** elements. The render method will only render a single root node inside it at a time. However, if you want to return multiple elements, the render method will require a '**div**' tag and put the entire content or elements inside it. This extra node to the DOM sometimes results in the wrong formatting of your HTML output and also not loved by the many developers.

Example

```
// Rendering with div tag
class App extends React.Component {
     render() {
      return (
         //Extraneous div element
```

```
        <div>
          <h2> Hello World! </h2>
          <p> Welcome to the JavaTpoint. </p>
        </div>
      );
    }
}
```

To solve this problem, React introduced **Fragments** from the **16.2** and above version. Fragments allow you to group a list of children without adding extra nodes to the DOM.

Syntax

```
<React.Fragment>
      <h2> child1 </h2>
    <p> child2 </p>
      .. ..... .... ...
</React.Fragment>
```

Example

```
// Rendering with fragments tag
class App extends React.Component {
    render() {
     return (
       <React.Fragment>
            <h2> Hello World! </h2>
        <p> Welcome to the JavaTpoint. </p>
         </React.Fragment>
     );
    }
}
```

Why we use Fragments?

The main reason to use Fragments tag is:

1. It makes the execution of code faster as compared to the div tag.
2. It takes less memory.

Fragments Short Syntax

There is also another shorthand exists for declaring fragments for the above method. It looks like **empty** tag in which we can use of '<>' and '' instead of the '**React.Fragment**'.

Example

```
//Rendering with short syntax
class Columns extends React.Component {
  render() {
    return (
      <>
        <h2> Hello World! </h2>
        <p> Welcome to the JavaTpoint </p>
      </>
    );
  }
}
```

Keyed Fragments

The shorthand syntax does not accept key attributes. You need a key for mapping a collection to an array of fragments such as to create a description list. If you need to provide keys, you have to declare the fragments with the explicit <**React.Fragment**> syntax.

Note: Key is the only attributes that can be passed with the Fragments.

Example

```
Function  = (props) {
  return (
    <Fragment>
      {props.items.data.map(item => (
        // Without the 'key', React will give a key warning
        <React.Fragment key={item.id}>
```

```
          <h2>{item.name}</h2>
          <p>{item.url}</p>
          <p>{item.description}</p>
        </React.Fragment>
      ))}
    </Fragment>
  )
}
```

Unit-28

React Router

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

Need of React Router

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

React Router Installation

React contains three different packages for routing. These are:

1. **react-router:** It provides the core routing components and functions for the React Router applications.

2. **react-router-native:** It is used for mobile applications.
3. **react-router-dom:** It is used for web applications design.

It is not possible to install react-router directly in your application. To use react routing, first, you need to install react-router-dom modules in your application. The below command is used to install react router dom.

```
$ npm install react-router-dom --save
```

Components in React Router

There are two types of router components:

- **<BrowserRouter>:** It is used for handling the dynamic URL.
- **<HashRouter>:** It is used for handling the static request.

Example

Step-1: In our project, we will create two more components along with **App.js**, which is already present.

About.js

```
import React from 'react'
class About extends React.Component {
  render() {
    return <h1>About</h1>
  }
}
export default About
```

Contact.js

```
import React from 'react'
class Contact extends React.Component {
  render() {
    return <h1>Contact</h1>
  }
}
export default Contact
```

App.js

```
import React from 'react'
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Home</h1>
      </div>
    )
  }
}
export default App
```

Step-2: For Routing, open the index.js file and import all the three component files in it. Here, you need to import line: **import { Route, Link, BrowserRouter as Router } from 'react-router-dom'** which helps us to implement the Routing. Now, our index.js file looks like below.

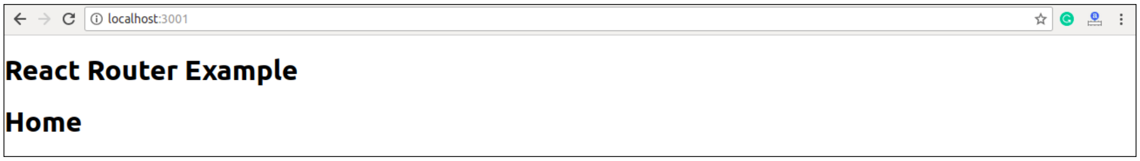
What is Route?

It is used to define and render component based on the specified path. It will accept components and render to define what should be rendered.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

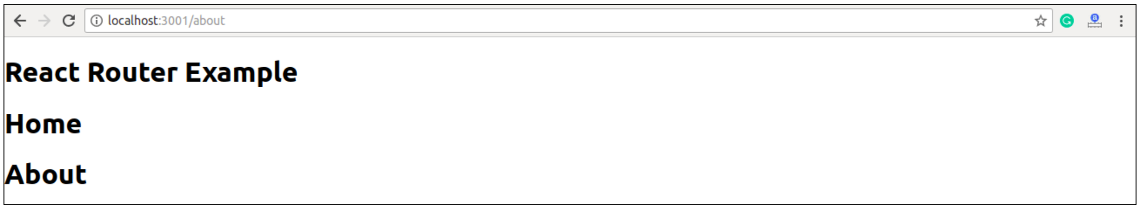
Step-3: Open **command prompt**, go to your project location, and then type **npm start**. You will get the following screen.


localhost:3001

React Router Example

Home

Now, if you enter **manually** in the browser: **localhost:3000/about**, you will see **About** component is rendered on the screen.


localhost:3001/about

React Router Example

Home

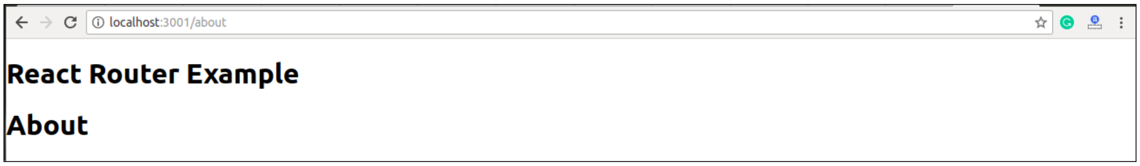
About

Step-4: In the above screen, you can see that **Home** component is still rendered. It is because the home path is '**/**' and about path is '**/about**', so you can observe that **slash** is common in both paths which render both components. To stop this behavior, you need to use the **exact** prop. It can be seen in the below example.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route exact path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Output



React Router Example

About

Adding Navigation using Link component

Sometimes, we want to need **multiple** links on a single page. When we click on any of that particular **Link**, it should load that page which is associated with that path without **reloading** the web page. To do this, we need to import **<Link>** component in the **index.js** file.

What is < Link> component?

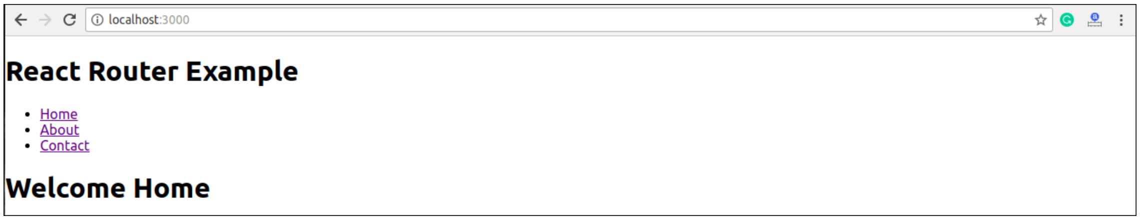
This component is used to create links which allow to **navigate** on different **URLs** and render its content without reloading the webpage.

Example

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <Link to="/">Home</Link>
        </li>
        <li>
          <Link to="/about">About</Link>
        </li>
        <li>
          <Link to="/contact">Contact</Link>
        </li>
      </ul>
      <Route exact path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Output

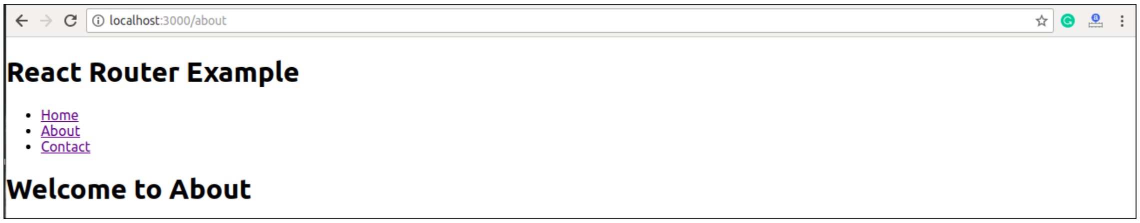


React Router Example

- Home
- About
- Contact

Welcome Home

After adding Link, you can see that the routes are rendered on the screen. Now, if you click on the **About**, you will see URL is changing and About component is rendered.



React Router Example

- Home
- About
- Contact

Welcome to About

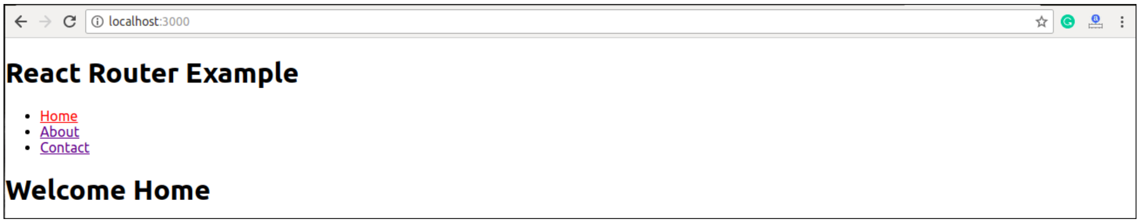
Now, we need to add some **styles** to the Link. So that when we click on any particular link, it can be easily **identified** which Link is **active**. To do this react router provides a new trick **NavLink** instead of **Link**. Now, in the index.js file, replace Link from Navlink and add properties **activeStyle**. The activeStyle properties mean when we click on the Link, it should have a specific style so that we can differentiate which one is currently active.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router, Route, Link, NavLink } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
```

```

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <NavLink to="/" exact activeStyle={
             {color:'red'}
          }>Home</NavLink>
        </li>
        <li>
          <NavLink to="/about" exact activeStyle={
             {color:'green'}
          }>About</NavLink>
        </li>
        <li>
          <NavLink to="/contact" exact activeStyle={
             {color:'magenta'}
          }>Contact</NavLink>
        </li>
      </ul>
      <Route exact path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Output

When we execute the above program, we will get the following screen in which we can see that **Home** link is of color **Red** and is the only currently **active** link.

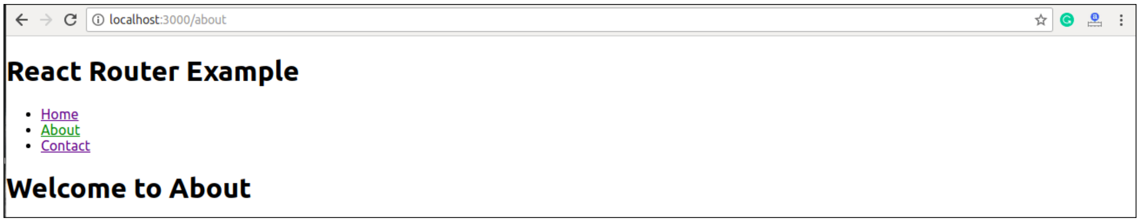


React Router Example

- Home
- About
- Contact

Welcome Home

Now, when we click on **About** link, its color shown **green** that is the currently **active** link.



localhost:3000/about

React Router Example

- Home
- About
- Contact

Welcome to About

<Link> vs <NavLink>

The Link component allows navigating the different routes on the websites, whereas NavLink component is used to add styles to the active routes.

React Router Switch

The <**Switch**> component is used to render components only when the path will be **matched**. Otherwise, it returns to the **not found** component.

To understand this, first, we need to create a **notfound** component.

notfound.js

```
import React from 'react'
const Notfound = () => <h1>Not found</h1>
export default Notfound
```

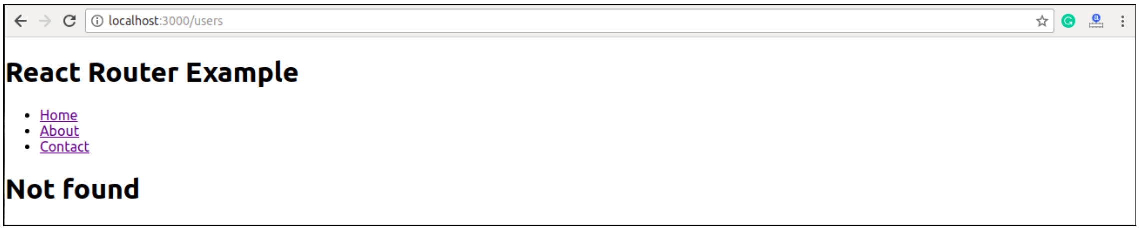
Now, import component in the index.js file. It can be seen in the below code.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router, Route, Link, NavLink, Switch } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
import Notfound from './notfound'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <NavLink to="/" exact activeStyle={
            {color:'red'}
          }>Home</NavLink>
        </li>
        <li>
          <NavLink to="/about" exact activeStyle={
            {color:'green'}
          }>About</NavLink>
        </li>
        <li>
          <NavLink to="/contact" exact activeStyle={
            {color:'magenta'}
          }>Contact</NavLink>
        </li>
      </ul>
      <Switch>
         <Route exact path="/" component={App} />
         <Route path="/about" component={About} />
         <Route path="/contact" component={Contact} />
         <Route component={Notfound} />
      </Switch>
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Output

If we manually enter the **wrong** path, it will give the not found error.



localhost:3000/users

React Router Example

- Home
- About
- Contact

Not found

React Router <Redirect>

A <Redirect> component is used to redirect to another route in our application to maintain the old URLs. It can be placed anywhere in the route hierarchy.

Nested Routing in React

Nested routing allows you to render **sub-routes** in your application. It can be understood in the below example.

Example

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router, Route, Link, NavLink, Switch } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
import Notfound from './notfound'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <NavLink to="/" exact activeStyle={
             {color:'red'}
          }>Home</NavLink>
        </li>
        <li>
          <NavLink to="/about" exact activeStyle={
             {color:'green'}
          }>About</NavLink>
        </li>
        <li>
          <NavLink to="/contact" exact activeStyle={
             {color:'magenta'}
          }>Contact</NavLink>
        </li>
      </ul>
      <Switch>
         <Route exact path="/" component={App} />
         <Route path="/about" component={About} />
         <Route path="/contact" component={Contact} />
        <Route component={Notfound} />
      </Switch>
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

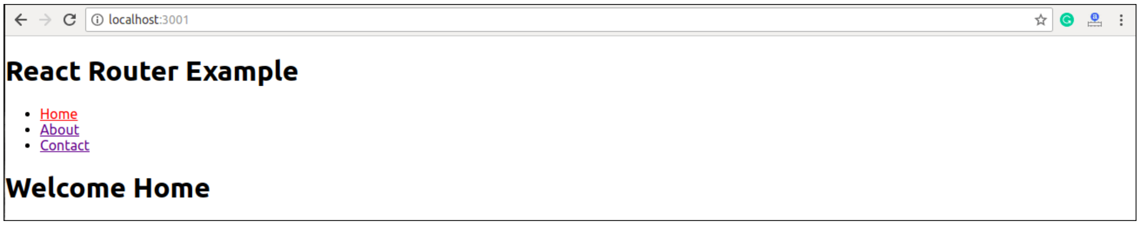
In the **contact.js** file, we need to import the **React Router** component to implement the **subroutes**.

contact.js

```
import React from 'react'
import { Route, Link } from 'react-router-dom'

const Contacts = ({ match }) => <p>{match.params.id}</p>

class Contact extends React.Component {
  render() {
    const { url } = this.props.match
    return (
      <div>
        <h1>Welcome to Contact Page</h1>
        <strong>Select contact Id</strong>
        <ul>
          <li>
            <Link to="/contact/1">Contacts 1 </Link>
          </li>
          <li>
            <Link to="/contact/2">Contacts 2 </Link>
          </li>
          <li>
            <Link to="/contact/3">Contacts 3 </Link>
          </li>
          <li>
            <Link to="/contact/4">Contacts 4 </Link>
          </li>
        </ul>
        <Route path="/contact/:id" component={Contacts} />
      </div>
    )
  }
}
export default Contact
```

Output

When we execute the above program, we will get the following output.

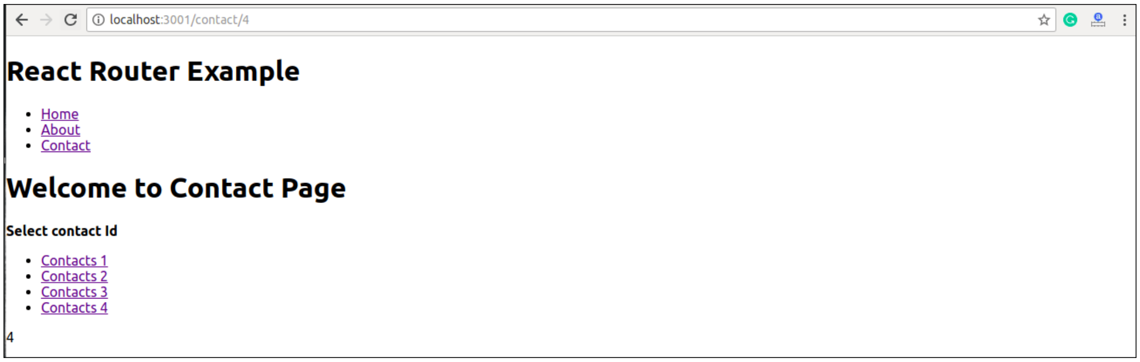


React Router Example

- Home
- About
- Contact

Welcome Home

After clicking the **Contact** link, we will get the contact list. Now, selecting any contact, we will get the corresponding output. It can be shown in the below example.



React Router Example

- Home
- About
- Contact

Welcome to Contact Page

Select contact Id

- Contacts 1
- Contacts 2
- Contacts 3
- Contacts 4

4

Benefits Of React Router

The benefits of React Router is given below:

- In this, it is not necessary to set the browser history manually.
- Link uses to navigate the internal links in the application. It is similar to the anchor tag.
- It uses Switch feature for rendering.
- The Router needs only a Single Child element.
- In this, every component is specified in .

Unit-29

React CSS

CSS in React is used to style the React App or Component. The **style** attribute is the most used attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in **camelCased** properties rather than a CSS string. There are many ways available to add styling to your React App or Component with CSS. Here, we are going to discuss mainly **four** ways to style React Components, which are given below:

1. Inline Styling
2. CSS Stylesheet
3. CSS Module
4. Styled Components

1. Inline Styling

The inline styles are specified with a JavaScript object in camelCase version of the style name. Its value is the style?s value, which we usually take in a string.

Example

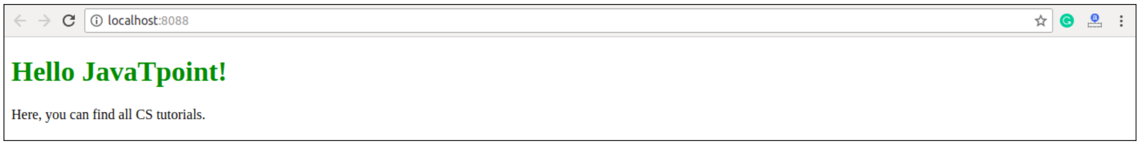
App.js

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

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "Green"}}>Hello JavaTpoint!</h1>
      <p>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

Note:

You can see in the above example, we have used two curly braces in:
<h1 style={{color: "Green"}}>Hello JavaTpoint!</h1>**.**
It is because, in JSX, JavaScript expressions are written inside curly braces, and JavaScript objects also use curly braces, so the above styling is written inside two sets of curly braces {{}}.

Output



Hello JavaTpoint!

Here, you can find all CS tutorials.

camelCase Property Name

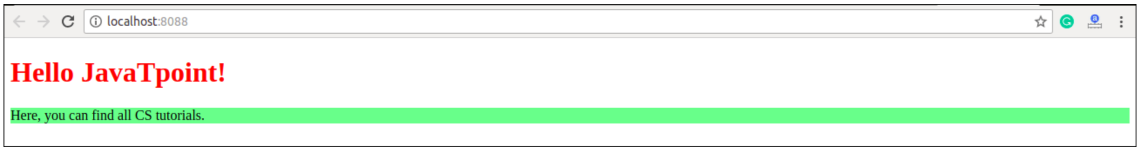
If the properties have two names, like **background-color**, it must be written in camel case syntax.

Example

App.js

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

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "Red"}}>Hello JavaTpoint!</h1>
      <p style={{backgroundColor: "lightgreen"}}>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

Output



Hello JavaTpoint!

Here, you can find all CS tutorials.

Using JavaScript Object

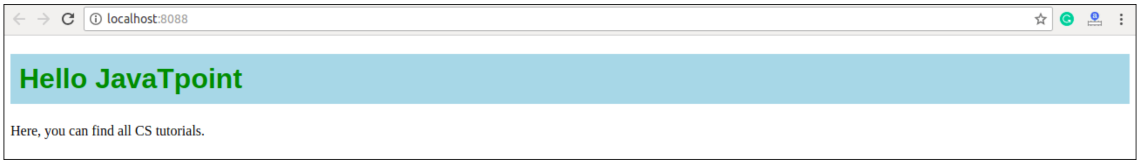
The inline styling also allows us to create an object with styling information and refer it in the style attribute.

Example

App.js

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

class App extends React.Component {
  render() {
    const mystyle = {
      color: "Green",
      backgroundColor: "lightBlue",
      padding: "10px",
      fontFamily: "Arial"
    };
    return (
      <div>
      <h1 style={mystyle}>Hello JavaTpoint</h1>
      <p>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

Output



Hello JavaTpoint

Here, you can find all CS tutorials.

2. CSS Stylesheet

You can write styling in a separate file for your React application, and save the file with a .css extension. Now, you can **import** this file in your application.

Example

App.js

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

class App extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello JavaTpoint</h1>
      <p>Here, you can find all CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

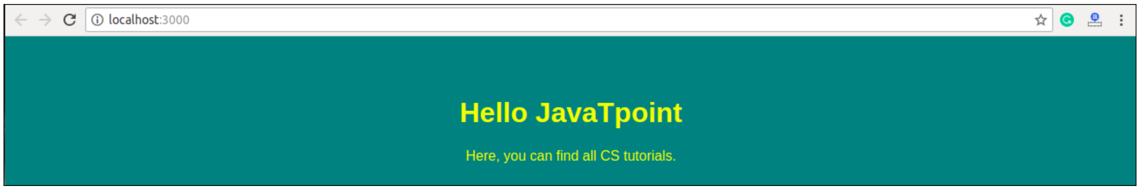
App.css

```
body {
  background-color: #008080;
  color: yellow;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport"
      content="width=device-width, initial-scale=1" />
    <title>React App</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

Output



Hello JavaTpoint

Here, you can find all CS tutorials.

3. CSS Module

CSS Module is another way of adding styles to your application. It is a **CSS file** where all class names and **animation** names are scoped locally by default. It is available only for the component which imports it, means any styling you add can never be applied to other components without your permission, and you never need to worry about name conflicts. You can create CSS Module with the **.module.css** extension like a **myStyles.module.css** name.

Example

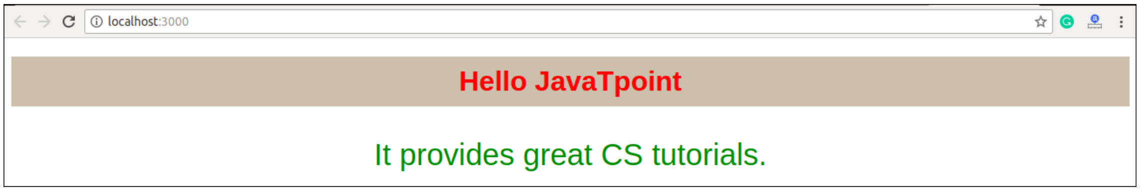
App.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import styles from './myStyles.module.css';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 className={styles.mystyle}>Hello JavaTpoint</h1>
      <p className={styles.parastyle}>It provides great CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

myStyles.module.css

```
.mystyle {
  background-color: #cdc0b0;
  color: Red;
  padding: 10px;
  font-family: Arial;
  text-align: center;
}

.parastyle{
  color: Green;
  font-family: Arial;
  font-size: 35px;
  text-align: center;
}
```

Output



```
Hello JavaTpoint

It provides great CS tutorials.
```

4. Styled Components

Styled-components is a **library** for React. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS.

The styled-components provides:

- Automatic critical CSS
- No class name bugs
- Easier deletion of CSS
- Simple dynamic styling
- Painless maintenance

Installation

The styled-components library takes a single command to install in your React application. which is:

```
$ npm install styled-components --save
```

Example

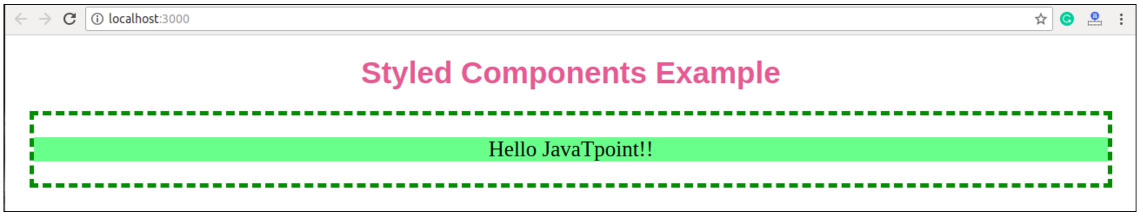
Here, we create a variable by selecting a particular HTML element such as **<div>**, **<Title>**, and **<paragraph>** where we store our style attributes. Now we can use the name of our variable as a wrapper **<Div></Div>** kind of React component.

App.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import styled from 'styled-components';

class App extends React.Component {
  render() {
    const Div:any = styled.div`
            margin: 20px;
            border: 5px dashed green;
            &:hover {
            background-color: ${(props:any) => props.hoverColor};
            }
            `;
    const Title = styled.h1`
            font-family: Arial;
            font-size: 35px;
            text-align: center;
            color: palevioletred;
            `;
    const Paragraph = styled.p`
            font-size: 25px;
            text-align: center;
            background-Color: lightgreen;
            `;
    return (
       <div>
            <Title>Styled Components Example</Title>
            <p></p>
            <Div hoverColor="Orange">
                 <Paragraph>Hello JavaTpoint!!</Paragraph>
            </Div>
        </div>
    );
  }
}
export default App;
```

Output

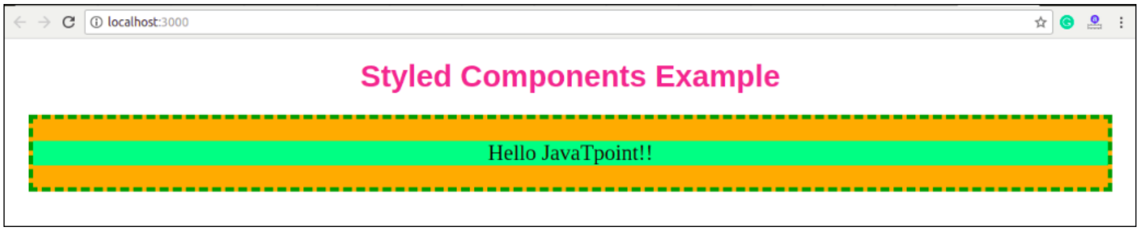
Now, execute the App.js file, we will get the output as shown below.



Styled Components Example

Hello JavaTpoint!!

When we move the mouse pointer over the image, its color will be changed, as shown in the below image.



Styled Components Example

Hello JavaTpoint!!

Unit-30

React Animation

The animation is a technique in which images are manipulated to appear as moving images. It is one of the most used technique to make an interactive web application. In React, we can add animation using an explicit group of components known as the **React Transition Group**.

React Transition Group is an add-on component for managing component states and useful for defining **entering** and **exiting** transitions. It is not able to animate styles by itself. Instead, it exposes transition states, manages classes and group elements, and manipulates the DOM in useful ways. It makes the implementation of visual transitions much easier.

React Transition group has mainly **two APIs** to create transitions. These are:

1. **ReactTransitionGroup:** It uses as a low-level API for animation.

2. **ReactCSSTransitionGroup:** It uses as a high-level API for implementing basic CSS transitions and animations.

Installation

We need to install **react-transition-group** for creating animation in React Web application. You can use the below command.

```
1. $ npm install react-transition-group --save
```

React Transition Group Components

React Transition Group API provides **three** main components. These are:

1. Transition
2. CSS Transition
3. Transition Group

Transition

It has a simple component API to describe a transition from one component state to another over time. It is mainly used to animate the **mounting** and **unmounting** of a component. It can also be used for in-place transition states as well.

We can access the Transition component into four states:

- entering
- entered
- exiting
- exited

CSSTransition

The CSSTransition component uses CSS stylesheet classes to write the transition and create animations. It is inspired by the **ng-animate** library. It can also inherit all the props of the transition component. We can divide the "CSSTransition" into **three** states. These are:

- Appear
- Enter
- Exit

CSSTransition component must be applied in a pair of class names to the child components. The first class is in the form of **name-stage** and the second class is in the **name-stage-active**. For example, you provide the name fade, and when it applies to the 'enter' stage, the two classes will be **fade-enter** and **fade-enter-active**. It may also take a prop as Timeout which defines the maximum time to animate.

TransitionGroup

This component is used to manage a set of transition components (Transition and CSSTransition) in a list. It is a state machine that controls the **mounting** and **unmounting** of components over time. The Transition component does not define any animation directly. Here, how 'list' item animates is based on the individual transition component. It means, the "TransitionGroup" component can have different animation within a component.

Let us see the example below, which clearly help to understand the React Animation.

Example

App.js

In the App.js file, import react-transition-group component, and create the CSSTransition component that uses as a wrapper of the component you want to animate.

We are going to use **transitionEnterTimeout** and **transitionLeaveTimeout**

for CSS Transition. The Enter and Leave animations used when we want to insert or delete elements from the list.

```
import React, { Component } from 'react';
import { CSSTransitionGroup } from 'react-transition-group';

class App extends React.Component {
    constructor(props) {
    super(props);
    this.state = {items: ['Blockchain', 'ReactJS', 'TypeScript', 'JavaTpoint']};
    this.handleAdd = this.handleAdd.bind(this);
  }

  handleAdd() {
    const newItems = this.state.items.concat([
      prompt('Enter Item Name')
    ]);
    this.setState({items: newItems});
  }

  handleRemove(i) {
    let newItems = this.state.items.slice();
    newItems.splice(i, 1);
    this.setState({items: newItems});
  }

  render() {
    const items = this.state.items.map((item, i) => (
      <div key={item} onClick={() => this.handleRemove(i)}>
        {item}
      </div>
    ));

    return (
      <div>
    <h1>Animation Example</h1>
            <button onClick={this.handleAdd}>Insert Item</button>
            <CSSTransitionGroup
               transitionName="example"
           transitionEnterTimeout={800}
               transitionLeaveTimeout={600}>
               {items}
            </CSSTransitionGroup>
      </div>
    );
  }
}
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'));
```

style.css

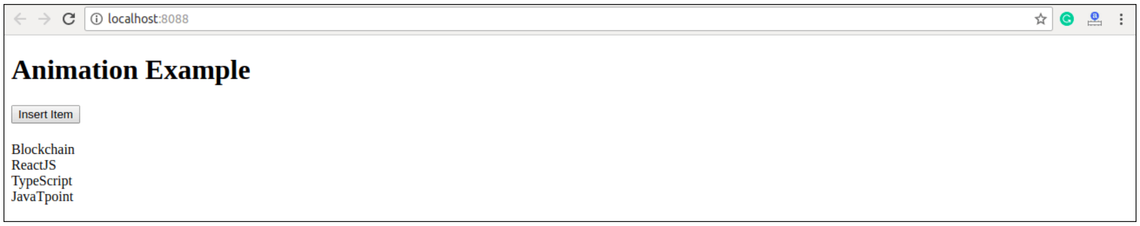
Add style.css file in your application, and add the following CSS styles. Now, to use this CSS file, you need to add the **link** of this file in your HTML file.

```
.example-enter {
  opacity: 0.01;
}

.example-enter.example-enter-active {
  opacity: 1;
  transition: opacity 500ms ease-in;
}

.example-leave {
  opacity: 1;
}

.example-leave.example-leave-active {
  opacity: 0.01;
  transition: opacity 300ms ease-in;
}
```

In the above example, the animation durations are specified in both the **CSS** and **render** method. It tells React component when to remove the animation classes from the list and if it is leaving when to remove the element from the DOM.

Output

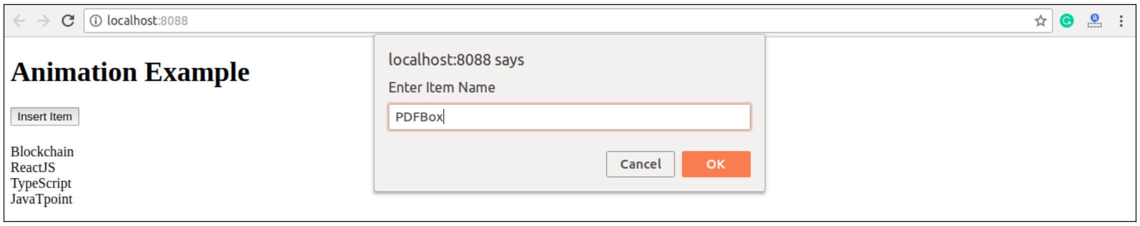
When we execute the above program, it gives the below output.



Animation Example

Blockchain
ReactJS
TypeScript
JavaTpoint

Click on '**Insert Item**' button, the following screen appears.



Animation Example

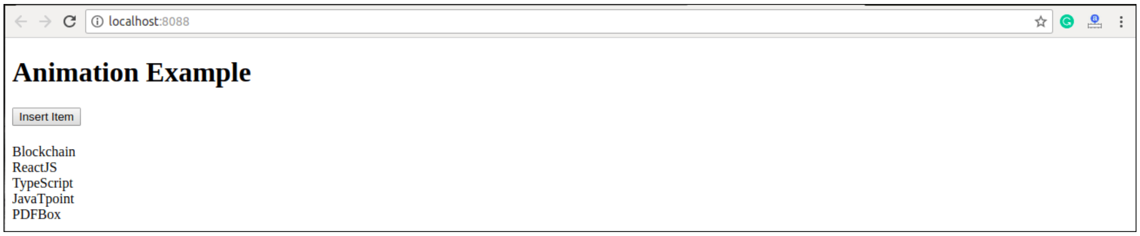
Blockchain
ReactJS
TypeScript
JavaTpoint

localhost:8088 says

Enter Item Name

PDFBox

Once we insert the item and press **Ok**, the new item can be added in the list with fade in style. Here, we can also delete any item from the list by clicking on the particular link.



Animation Example

Blockchain
ReactJS
TypeScript
JavaTpoint
PDFBox

Unit-31

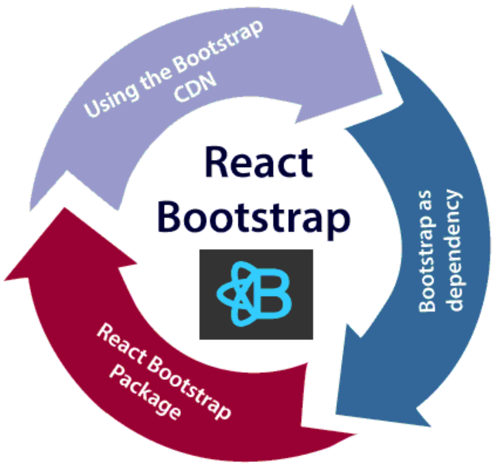
React Bootstrap

Single-page applications gaining popularity over the last few years, so many front-end frameworks have introduced such as Angular, React, Vue.js, Ember, etc. As a result, jQuery is not a necessary requirement for building web apps. Today, react has the most used JavaScript framework for building web applications, and Bootstrap become the most popular CSS framework. So, it is necessary to learn various ways in which Bootstrap can be used in React apps, which is the main aim of this section.

Adding Bootstrap for React

We can add Bootstrap to the React app in several ways. The **three** most common ways are given below:

1. Using the Bootstrap CDN
2. Bootstrap as Dependency
3. React Bootstrap Package



Using the Bootstrap CDN

It is the easiest way of adding Bootstrap to the React app. There is no need to install or download Bootstrap. We can simply put an **<link>** into the **<head>** section of the **index.html** file of the React app as shown in the following snippet.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
```

If there is a need to use Bootstrap components which depend on JavaScript/jQuery in the React application, we need to include **jQuery**, **Popper.js**, and **Bootstrap.js** in the document. Add the following imports in the **<script>** tags near the end of the closing **</body>** tag of the **index.html** file.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
```

```
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
```

In the above snippet, we have used jQuery's slim version, although we can also use the full version as well. Now, Bootstrap is successfully added in the React application, and we can use all the CSS utilities and UI components available from Bootstrap in the React application.

Bootstrap as Dependency

If we are using a build tool or a module bundler such as Webpack, then importing Bootstrap as dependency is the preferred option for adding Bootstrap to the React application. We can install Bootstrap as a dependency for the React app. To install the Bootstrap, run the following commands in the terminal window.

```
$ npm install bootstrap --save
```

Once Bootstrap is installed, we can import it in the React application entry file. If the React project created using the **create-react-app** tool, open the **src/index.js** file, and add the following code:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Now, we can use the CSS classes and utilities in the React application. Also, if we want to use the JavaScript components, we need to install the **jquery** and **popper.js** packages from **npm**. To install the following packages, run the following command in the terminal window.

```
$ npm install jquery popper.js
```

Next, go to the **src/index.js** file and add the following imports.

```
import $ from 'jquery';
import Popper from 'popper.js';
import 'bootstrap/dist/js/bootstrap.bundle.min';
```

Now, we can use Bootstrap JavaScript Components in the React application.

React Bootstrap Package

The React Bootstrap package is the most popular way to add bootstrap in the React application. There are many Bootstrap packages built by the community, which aim to rebuild Bootstrap components as React components. The **two** most popular Bootstrap packages are:

1. **react-bootstrap:** It is a complete re-implementation of the Bootstrap components as React components. It does not need any dependencies like bootstrap.js or jQuery. If the React setup and React-Bootstrap installed, we have everything which we need.
2. **reactstrap:** It is a library which contains React Bootstrap 4 components that favor composition and control. It does not depend on jQuery or Bootstrap JavaScript. However, react-popper is needed for advanced positioning of content such as Tooltips, Popovers, and auto-flipping Dropdowns.

React Bootstrap Installation

Let us create a new React app using the **create-react-app** command as follows.

```
$ npx create-react-app react-bootstrap-app
```

After creating the React app, the best way to install Bootstrap is via the npm package. To install Bootstrap, navigate to the React app folder, and run the following command.

```
$ npm install react-bootstrap bootstrap --save
```

Importing Bootstrap

Now, open the **src/index.js** file and add the following code to import the Bootstrap file.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

We can also import individual components **like import { SplitButton, Dropdown } from 'react-bootstrap';** instead of the entire library. It provides the specific components which we need to use, and can significantly reduce the amount of code.

In the React app, create a new file named **ThemeSwitcher.js** in the **src** directory and put the following code.

```
import React, { Component } from 'react';
import { SplitButton, Dropdown } from 'react-bootstrap';

class ThemeSwitcher extends Component {

  state = { theme: null }

  chooseTheme = (theme, evt) => {
    evt.preventDefault();
    if (theme.toLowerCase() === 'reset') { theme = null }
    this.setState({ theme });
  }

  render() {
    const { theme } = this.state;
    const themeClass = theme ? theme.toLowerCase() : 'default';

    const parentContainerStyles = {
      position: 'absolute',
      height: '100%',
      width: '100%',
      display: 'table'
    };

    const subContainerStyles = {
      position: 'relative',
      height: '100%',
      width: '100%',
      display: 'table-cell',
    };

    return (
      <div style={parentContainerStyles}>
        <div style={subContainerStyles}>
```

```

     <span className={`h1 center-block text-center text-${theme ? themeClass : 'muted'}`} style={{ marginBottom: 25 }}>{theme || 'Default' }</span>

     <div className="center-block text-center">
      <SplitButton bsSize="large" bsStyle={themeClass} title={`${theme || 'Default Block'} Theme`}>
       <Dropdown.Item eventKey="Primary Block" onSelect={this.chooseTheme}>Primary Theme</Dropdown.Item>
       <Dropdown.Item eventKey="Danger Block" onSelect={this.chooseTheme}>Danger Theme</Dropdown.Item>
       <Dropdown.Item eventKey="Success Block" onSelect={this.chooseTheme}>Success Theme</Dropdown.Item>
       <Dropdown.Item divider />
       <Dropdown.Item eventKey="Reset Block" onSelect={this.chooseTheme}>Default Theme</Dropdown.Item>
      </SplitButton>
     </div>
    </div>
   </div>
  );
 }
}
export default ThemeSwitcher;
```

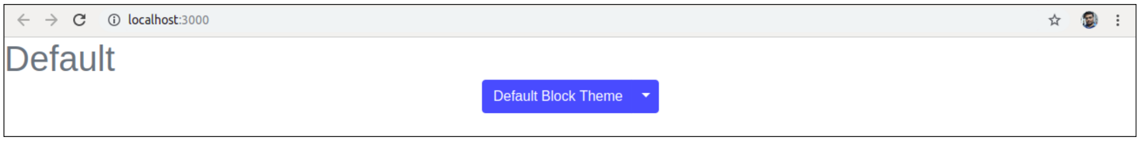
Now, update the **src/index.js** file with the following snippet.

Index.js

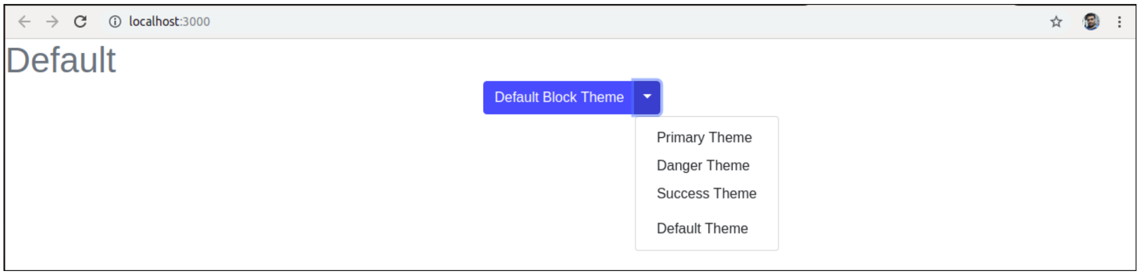
```
import 'bootstrap/dist/css/bootstrap.min.css';
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';
import './index.css';
import ThemeSwitcher from './ThemeSwitcher';

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

Output

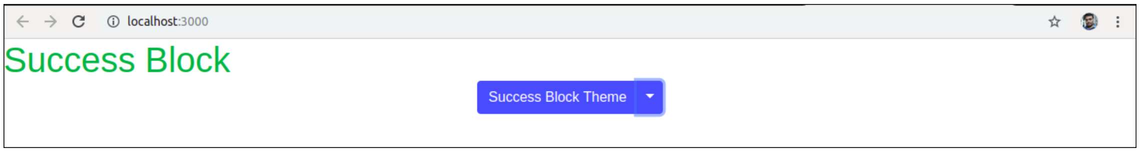
When we execute the React app, we should get the output as below.



Click on the dropdown menu. We will get the following screen.



Now, if we choose the **Success Theme**, we will get the below screen.



Using reactstrap

Let us create a new React app using the create-react-app command as follows.

```
$ npx create-react-app reactstrap-app
```

Next, install the **reactstrap** via the npm package. To install reactstrap, navigate to the React app folder, and run the following command.

```
$ npm install bootstrap reactstrap --save
```

Importing Bootstrap

Now, open the **src/index.js** file and add the following code to import the Bootstrap file.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

We can also import individual components **like import { Button, Dropdown } from 'reactstrap';** instead of the entire library. It provides the specific components which we need to use, and can significantly reduce the amount of code.

In the React app, create a new file named **ThemeSwitcher.js** in the **src** directory and put the following code.

```
import React, { Component } from 'react';
import { Button, ButtonDropdown, DropdownToggle, DropdownMenu, DropdownItem } from 'reactstrap';

class ThemeSwitcher extends Component {

  state = { theme: null, dropdownOpen: false }

  toggleDropdown = () => {
    this.setState({ dropdownOpen: !this.state.dropdownOpen });
  }

  resetTheme = evt => {
    evt.preventDefault();
    this.setState({ theme: null });
  }

  chooseTheme = (theme, evt) => {
    evt.preventDefault();
    this.setState({ theme });
  }
  render() {
    const { theme, dropdownOpen } = this.state;
    const themeClass = theme ? theme.toLowerCase() : 'secondary';

    return (
      <div className="d-flex flex-wrap justify-content-center align-items-center">

        <span className={`h1 mb-4 w-100 text-center text-${themeClass}`}>{theme || 'Default'}</span>

        <ButtonDropdown isOpen={dropdownOpen} toggle={this.toggleDropdown}>
          <Button id="caret" color={themeClass}>{theme || 'Custom'} Theme</Button>
          <DropdownToggle caret size="lg" color={themeClass} />
          <DropdownMenu>
            <DropdownItem onClick={e => this.chooseTheme('Primary', e)}>Primary Theme</DropdownItem>
            <DropdownItem onClick={e => this.chooseTheme('Danger', e)}>Danger Theme</DropdownItem>
            <DropdownItem onClick={e => this.chooseTheme('Success', e)}>Success Theme</DropdownItem>
            <DropdownItem divider />
            <DropdownItem onClick={this.resetTheme}>Default Theme</DropdownItem>

          </DropdownMenu>
        </ButtonDropdown>

      </div>
    );
  }
}
export default ThemeSwitcher;
```

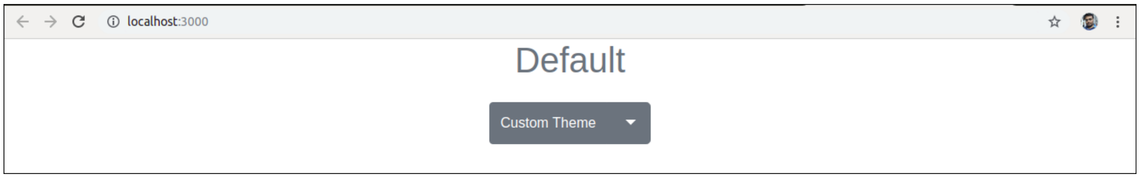
Now, update the **src/index.js** file with the following snippet.

Index.js

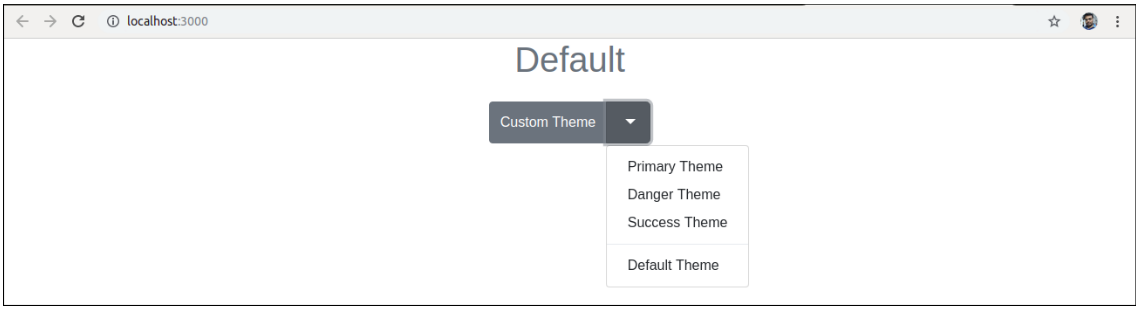
```
import 'bootstrap/dist/css/bootstrap.min.css';
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';
import './index.css';
import ThemeSwitcher from './ThemeSwitcher';

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

Output

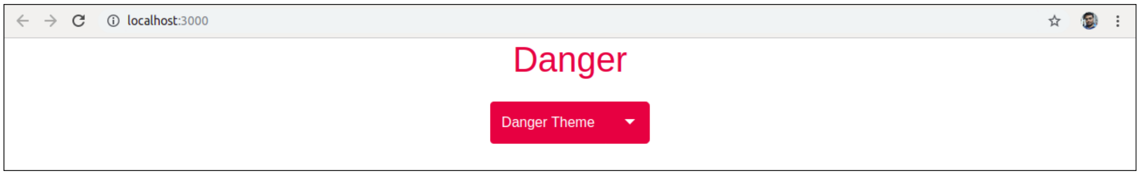
When we execute the React app, we should get the output as below.



Click on the dropdown menu. We will get the following screen.



Now, if we choose the **Danger Theme**, we will get the below screen.



Unit-32

React Map

A map is a data collection type where data is stored in the form of pairs. It contains a unique key. The value stored in the map must be mapped to the key. We cannot store a duplicate pair in the map(). It is because of the uniqueness of each stored key. It is mainly used for fast searching and looking up data.

In React, the ?map? method used to traverse and display a list of similar objects of a component. A map is not the feature of React. Instead, it is the standard JavaScript function that could be called on any array. The map() method creates a new array by calling a provided function on every element in the calling array.

Example

In the given example, the map() function takes an array of numbers and double their values. We assign the new array returned by map() to the variable doubleValue and log it.

```
var numbers = [1, 2, 3, 4, 5];
const doubleValue = numbers.map((number)=>{
    return (number * 2);
});
console.log(doubleValue);
```

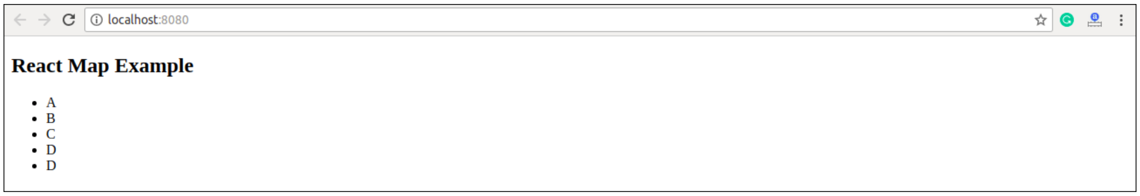
In React, the map() method used for:

1. Traversing the list element.

Example

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

function NameList(props) {
  const myLists = props.myLists;
  const listItems = myLists.map((myList) =>
    <li>{myList}</li>
  );
  return (
    <div>
          <h2>React Map Example</h2>
              <ul>{listItems}</ul>
    </div>
  );
}
const myLists = ['A', 'B', 'C', 'D', 'D'];
ReactDOM.render(
  <NameList myLists={myLists} />,
  document.getElementById('app')
);
export default App;
```

Output



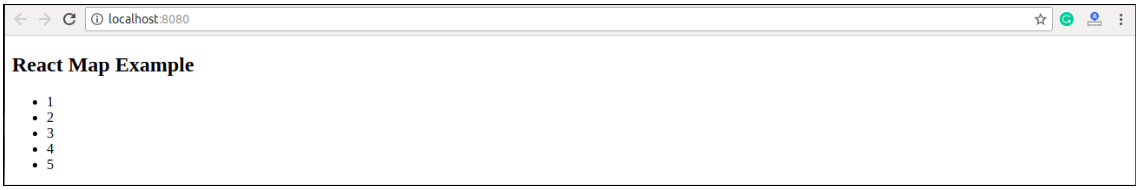
```
React Map Example

• A
• B
• C
• D
• D
```

2. Traversing the list element with keys.

Example

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

function ListItem(props) {
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const numbers = props.numbers;
const listItems = numbers.map((number) =>
  <ListItem key={number.toString()}
           value={number} />
);
return (
  <div>
    <h2>React Map Example</h2>
        <ul> {listItems} </ul>
  </div>
);
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('app')
);
export default App;
```

Output



```
React Map Example

  • 1
  • 2
  • 3
  • 4
  • 5
```

Unit-33

React Table

A table is an arrangement which organizes information into rows and columns. It is used to store and display data in a structured format.

The react-table is a lightweight, fast, fully customizable (JSX, templates, state, styles, callbacks), and extendable Datagrid built for React. It is fully controllable via optional props and callbacks.

Features

1. It is lightweight at 11kb (and only need 2kb more for styles).
2. It is fully customizable (JSX, templates, state, styles, callbacks).
3. It is fully controllable via optional props and callbacks.
4. It has client-side & Server-side pagination.
5. It has filters.
6. Pivoting & Aggregation
7. Minimal design & easily themeable

Installation

Let us create a **React app** using the following command.

```
$ npx create-react-app myreactapp
```

Next, we need to install **react-table**. We can install react-table via npm command, which is given below.

```
$ npm install react-table
```

Once, we have installed react-table, we need to **import** the react-table into the react component. To do this, open the **src/App.js** file and add the following snippet.

```
import ReactTable from "react-table";
```

Let us assume we have data which needs to be rendered using react-table.

```
const data = [{
        name: 'Ayaan',
        age: 26
        },{
        name: 'Ahana',
        age: 22
        },{
        name: 'Peter',
        age: 40
        },{
        name: 'Virat',
        age: 30
        },{
        name: 'Rohit',
        age: 32
        },{
        name: 'Dhoni',
        age: 37
        }]
```

Along with data, we also need to specify the **column info** with **column attributes**.

```
const columns = [{
       Header: 'Name',
       accessor: 'name'
      },{
      Header: 'Age',
      accessor: 'age'
      }]
```

Inside the render method, we need to bind this data with react-table and then returns the react-table.

```
return (
     <div>
        <ReactTable
            data={data}
            columns={columns}
            defaultPageSize = {2}
            pageSizeOptions = {[2,4, 6]}
         />
     </div>
)
```

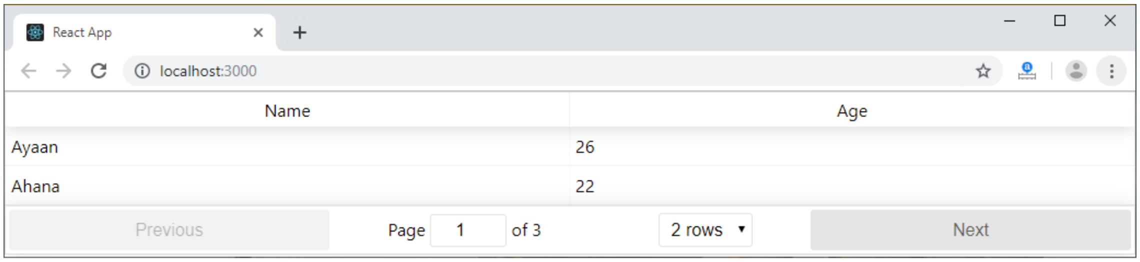
Now, our **src/App.js** file looks like as below.

```
import React, { Component } from 'react';
import ReactTable from "react-table";
import "react-table/react-table.css";

class App extends Component {
  render() {
     const data = [{
        name: 'Ayaan',
        age: 26
        },{
```

```
        name: 'Ahana',
        age: 22
        },{
        name: 'Peter',
        age: 40
        },{
        name: 'Virat',
        age: 30
        },{
        name: 'Rohit',
        age: 32
        },{
        name: 'Dhoni',
        age: 37
        }]
    const columns = [{
      Header: 'Name',
      accessor: 'name'
      },{
      Header: 'Age',
      accessor: 'age'
      }]
    return (
        <div>
          <ReactTable
            data={data}
            columns={columns}
            defaultPageSize = {2}
            pageSizeOptions = {[2,4, 6]}
          />
        </div>
    )
  }
}
export default App;
```

Output

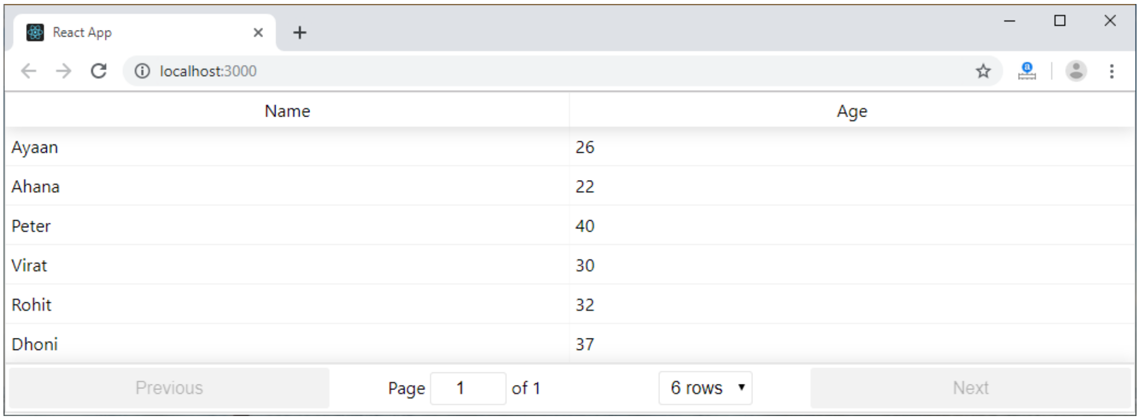
When we execute the React app, we will get the output as below.



Name	Age
Ayaan	26
Ahana	22

Page 1 of 3 2 rows

Now, change the rows dropdown menu, we will get the output as below.



Name	Age
Ayaan	26
Ahana	22
Peter	40
Virat	30
Rohit	32
Dhoni	37

Page 1 of 1 6 rows

Unit-34

React Higher-Order Components

It is also known as HOC. In React, HOC is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component. According to the official website, it is not the feature(part) in React API, but a pattern that emerges from React compositional nature. They are similar to JavaScript functions used for adding additional functionalities to the existing component.

A higher order component function accepts another function as an argument. The **map** function is the best example to understand this. The main goal of this is to decompose the component logic into simpler and smaller functions that can be reused as you need.

Syntax

```
1. const NewComponent = higherOrderComponent(WrappedComponent);
```

We know that component transforms props into UI, and a higher-order component converts a component another component and allows to add additional data or functionality into this. **Hocs** are common in **third-party** libraries. The examples of HOCs are **Redux's connect** and **Relay's createFragmentContainer**.

Now, we can understand the **working of HOCs** from the below example.

```
//Function Creation
function add (a, b) {
  return a + b
}
function higherOrder(a, addReference) {
  return addReference(a, 20)
}
//Function call
higherOrder(30, add) // 50
```

In the above example, we have created two functions **add()** and **higherOrder()**. Now, we provide the add() function as an **argument** to the higherOrder() function. For invoking, rename it **addReference** in the higherOrder() function, and then **invoke it**.

Here, the function you are passing is called a callback function, and the function where you are passing the callback function is called a **higher-order(HOCs)** function.

Example

Create a new file with the name HOC.js. In this file, we have made one function HOC. It accepts one **argument** as a component. Here, that component is **App**.

HOC.js

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

export default function Hoc(HocComponent){
    return class extends Component{
        render(){
            return (
                <div>
                    <HocComponent></HocComponent>
            </div>

          );
        }
    }
}
```

Now, include **HOC.js** file into the **App.js** file. In this file, we need to **call** the HOC function.

```
App = Hoc(App);
```

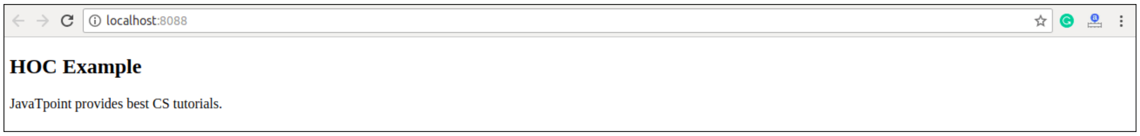
The App component wrapped inside another React component so that we can modify it. Thus, it becomes the primary application of the Higher-Order Components.

App.js

```
import React, { Component } from 'react';
import Hoc from './HOC';

class App extends Component {
  render() {
    return (
      <div>
        <h2>HOC Example</h2>
        JavaTpoint provides best CS tutorials.
      </div>
    )
  }
}
App = Hoc(App);
export default App;
```

Output

When we execute the above file, it will give the output as below screen.



HOC Example

JavaTpoint provides best CS tutorials.

Higher-Order Component Conventions

- Do not use HOCs inside the render method of a component.
- The static methods must be copied over to have access to them. You can do this using hoist-non-react-statics package to automatically copy all non-React static methods.
- HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument. If you add a ref to an element in the HOC component, the ref refers to an instance of the outermost container component, not the wrapped component.

Unit-35

React Code Splitting

The React app bundled their files using tools like **Webpack** or **Browserfy**. Bundling is a process which takes multiple files and merges them into a single file, which is called a **bundle**. The bundle is responsible for loading an entire app at once on the webpage. We can understand it from the below example.

App.js

```
import { add } from './math.js';

console.log(add(16, 26)); // 42
```

math.js

```
export function add(a, b) {
  return a + b;
}
```

Bundle file as like below:

```
function add(a, b) {
  return a + b;
}

console.log(add(16, 26)); // 42
```

As our app grows, our bundle will grow too, especially when we are using large third-party libraries. If the bundle size gets large, it takes a long time to load on a webpage. For avoiding the large bundling, it?s good to start ?splitting? your bundle.

React 16.6.0, released in **October 2018**, and introduced a way of performing code splitting. Code-Splitting is a feature supported by Webpack and Browserify, which can create multiple bundles that can be dynamically loaded at runtime.

Code splitting uses **React.lazy** and **Suspense** tool/library, which helps you to load a dependency lazily and only load it when needed by the user.

The code splitting improves:

- The performance of the app
- The impact on memory
- The downloaded Kilobytes (or Megabytes) size

React.lazy

The best way for code splitting into the app is through the dynamic **import()** syntax. The React.lazy function allows us to render a dynamic import as a regular component.

Before

```
import ExampleComponent from './ExampleComponent';

function MyComponent() {
  return (
    <div>
      <ExampleComponent />
    </div>
  );
}
```

After

```
const ExampleComponent = React.lazy(() => import('./ExampleComponent'));

function MyComponent() {
  return (
    <div>
      <ExampleComponent />
    </div>
  );
}
```

The above code snippet automatically loads the bundle which contains the ExampleComponent when the ExampleComponent gets rendered.

Suspense

If the module which contains the ExampleComponent is not yet loaded by the function component(MyComponent), then we need to show some **fallback** content while we are waiting for it to load. We can do this using the suspense component. In other words, the suspense component is responsible for handling the output when the lazy component is fetched and rendered.

```
const ExampleComponent = React.lazy(() => import('./ ExampleComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <ExampleComponent />
      </Suspense>
    </div>
  );
}
```

The **fallback** prop accepts the React elements which you want to render while waiting for the component to load. We can combine multiple lazy components with a single Suspense component. It can be seen in the below example.

```
const ExampleComponent = React.lazy(() => import('./ ExampleComponent'));
const ExamComponent = React.lazy(() => import('./ ExamComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
```

```
          <ExampleComponent />
          <ExamComponent />
        </section>
      </Suspense>
    </div>
  );
}
```

Note: **React.lazy and Suspense components are not yet available for server-side rendering. For code-splitting in a server-rendered app, it is recommended to use Loadable Components.**

Error boundaries

If any module fails to load, for example, due to network failure, we will get an error. We can handle these errors with Error Boundaries. Once we have created the Error Boundary, we can use it anywhere above our lazy components to display an error state.

```
import MyErrorBoundary from './MyErrorBoundary';
const ExampleComponent = React.lazy(() => import('./ ExampleComponent'));
const ExamComponent = React.lazy(() => import('./ ExamComponent'));

const MyComponent = () => (
  <div>
    <MyErrorBoundary>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
          <ExampleComponent />
          <ExamComponent />
        </section>
      </Suspense>
    </MyErrorBoundary>
  </div>
);
```

Route-based code splitting

It is very tricky to decide where we introduce code splitting in the app. For this, we have to make sure that we choose the place which will split the bundles evenly without disrupting the user experience.

The route is the best place to start the code splitting. Route based code splitting is essential during the page transitions on the web, which takes some amount of time to load. Here is an example of how to setup route-based code splitting into the app using React Router with React.lazy.

```
import { Switch, BrowserRouter as Router, Route} from 'react-router-dom';
import React, { Suspense, lazy } from 'react';

const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About'));
const Contact = lazy(() => import('./routes/Contact'));

const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Switch>
        <Route exact path="/" component={Home}/>
        <Route path="/about" component={About}/>
        <Route path="/contact" component={Contact}/>
      </Switch>
    </Suspense>
  </Router>
);
```

Named Export

Currently, React.lazy supports default exports only. If any module you want to import using named exports, you need to create an intermediate module that re-exports it as the default. We can understand it from the below example.

ExampleComponents.js

```
export const MyFirstComponent = /* ... */;
export const MySecondComponent = /* ... */;
```

MyFirstComponent.js

```
export { MyFirstComponent as default } from "./ExampleComponents.js";
```

MyApp.js

```
import React, { lazy } from 'react';
const MyFirstComponent = lazy(() => import("./MyFirstComponent.js"));
```

Unit-36

React Context

Context allows passing data through the component tree without passing props down manually at every level.

In React application, we passed data in a top-down approach via props. Sometimes it is inconvenient for certain types of props that are required by many components in the React application. Context provides a way to pass values between components without explicitly passing a prop through every level of the component tree.

How to use Context

There are two main steps to use the React context into the React application:

1. Setup a context provider and define the data which you want to store.
2. Use a context consumer whenever you need the data from the store

When to use Context

Context is used to share data which can be considered "global" for React components tree and use that data where needed, such as the current authenticated user, theme, etc. For example, in the below code snippet, we manually thread through a "theme" prop to style the Button component.

```
class App extends React.Component {
  render() {
    return <Toolbar theme="dark" />;
  }
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton theme={props.theme} />
    </div>
  );
}

```

```
class ThemedButton extends React.Component {
  render() {
    return <Button theme={this.props.theme} />;
  }
}
```

In the above code, the Toolbar function component takes an extra "theme" prop and pass it to the Themed Button. It can become inconvenient if every single button in the app needs to know the theme because it would be required to pass through all components. But using context, we can avoid passing props for every component through intermediate elements.

We can understand it from the below example. Here, context passes a value into the component tree without explicitly threading it through every single component.

```
// Create a context for the current theme which is "light" as the default.
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    /* Use a ContextProvider to pass the current theme, which allows every component to read it, no matter how deep it is. Here, we are passing the "dark" theme as the current value.*/

    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

// Now, it is not required to pass the theme down explicitly for every component.
function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

```

```
class ThemedButton extends React.Component {
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

React Context API

The React Context API is a component structure, which allows us to share data across all levels of the application. The main aim of Context API is to solve the problem of prop drilling (also called "Threading"). The Context API in React are given below.

1. React.createContext
2. Context.provider
3. Context.Consumer
4. Class.contextType

React.createContext

It creates a context object. When React renders a component which subscribes to this context object, then it will read the current context value from the matching provider in the component tree.

Syntax

```
const MyContext = React.createContext(defaultValue);
```

When a component does not have a matching Provider in the component tree, it returns the defaultValue argument. It is very helpful for testing components isolation (separately) without wrapping them.

Context.Provider

Every Context object has a Provider React component which allows consuming components to subscribe to context changes. It acts as a delivery service. When a consumer component asks for something, it finds it in the context and provides it to where it is needed.

Syntax

```
<MyContext.Provider value={/* some value */}>
```

It accepts the value prop and passes to consuming components which are descendants of this Provider. We can connect one Provider with many consumers. Context Providers can be nested to override values deeper within the component tree. All consumers that are descendants of a Provider always re-render whenever the Provider's value prop is changed. The changes are determined by comparing the old and new values using the same algorithm as **Object.is** algorithm.

Context.Consumer

It is the React component which subscribes to the context changes. It allows us to subscribe to the context within the function component. It requires the function as a component. A consumer is used to request data through the provider and manipulate the central data store when the provider allows it.

Syntax

```
<MyContext.Consumer>
      {value => /* render something which is based on the context value */}
</MyContext.Consumer>
```

The function component receives the current context value and then returns a React node. The value argument which passed to the function will be equal to the value prop of the closest Provider for this context in the component tree. If there is no Provider for this context, the value argument will be equal to the defaultValue which was passed to createContext().

Class.contextType

The contextType property on a class used to assign a Context object which is created by React.createContext(). It allows you to consume the closest current value of that Context type using this.context. We can reference this in any of the component life-cycle methods, including the render function.

Note: **We can only subscribe to a single context using this API. If we want to use the experimental public class field's syntax, we can use a static class field to initialize the contextType.**

React Context API Example

Step1 Create a new React app using the following command.

```
$ npx create-react-app mycontextapi
```

Step2 Install bootstrap CSS framework using the following command.

```
$ npm install react-bootstrap bootstrap --save
```

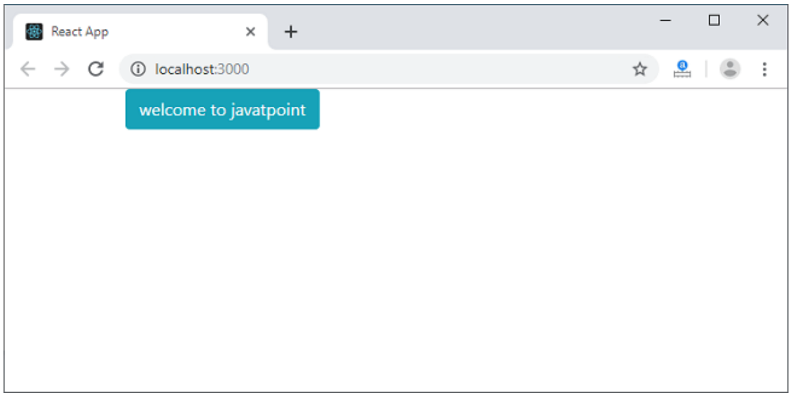
Step3 Add the following code snippet in the src/APP.js file.

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';

const BtnColorContext = React.createContext('btn btn-darkyellow');

class App extends Component {
  render() {
    return (
      <BtnColorContext.Provider value="btn btn-info">
        <Button />
      </BtnColorContext.Provider>
    );
  }
}

function Button(props) {
  return (
  <div className="container">
    <ThemedButton />
  </div>
  );
}

class ThemedButton extends Component {

  static contextType = BtnColorContext;
  render() {
    return <button className={this.context} >
      welcome to javatpoint
    </button>;
  }
}
export default App;
```

In the above code snippet, we have created the context using React.createContext(), which returns the Context object. After that, we have created the wrapper component which returns the Provider component, and then add all the elements as children from which we want to access the context.

output:

When we run the React app, we will get the following screen.



unit-37

React Hooks

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.

When to use a Hooks

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

Pre-requisites for React Hooks

1. Node version 6 or above
2. NPM version 5.2 or above
3. Create-react-app tool for running the React App

React Hooks Installation

To use React Hooks, we need to run the following commands:

```
$ npm install react@16.8.0-alpha.1 --save
$ npm install react-dom@16.8.0-alpha.1 --save
```

The above command will install the latest React and React-DOM alpha versions which support React Hooks. Make sure the **package.json** file lists the React and React-DOM dependencies as given below.

```
"react": "^16.8.0-alpha.1",
"react-dom": "^16.8.0-alpha.1",
```

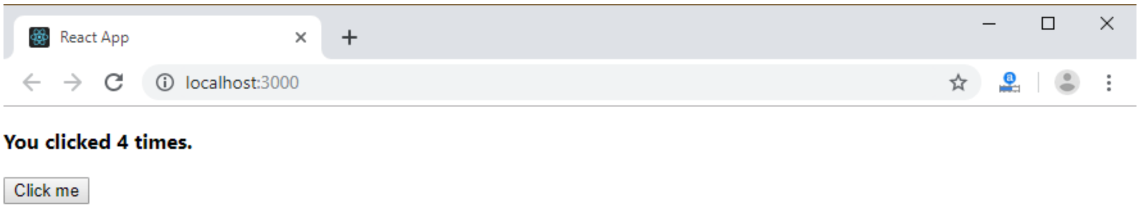
Hooks State

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrieving state. Let us understand Hook state with the following example.

App.js

```
import React, { useState } from 'react';

function CountApp() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
export default CountApp;
```

output:



You clicked 4 times.

In the above example, useState is the Hook which needs to call inside a function component to add some local state to it. The useState returns a pair where the first element is the current state value/initial value, and the second one is a function which allows us to update it. After that, we will call this function from an event handler or somewhere else. The useState is similar to this.setState in class. The equivalent code without Hooks looks like as below.

App.js

```
import React, { useState } from 'react';

class CountApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  render() {
    return (
      <div>
        <p><b>You clicked {this.state.count} times</b></p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
export default CountApp;
```

Hooks Effect

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

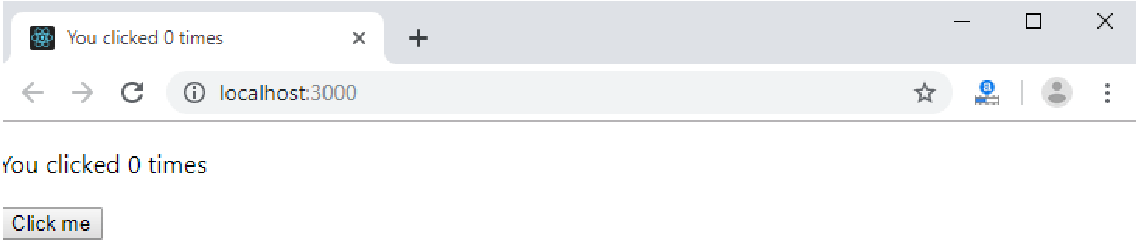
- Updating the DOM,
- Fetching and consuming data from a server API,
- Setting up a subscription, etc.

Let us understand Hook Effect with the following example.

```
import React, { useState, useEffect } from 'react';

function CounterExample() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
export default CounterExample;
```

The above code is based on the previous example with a new feature which we set the document title to a custom message, including the number of clicks.

Output:



You clicked 0 times

In React component, there are two types of side effects:

1. Effects Without Cleanup
2. Effects With Cleanup

Effects without Cleanup

It is used in useEffect which does not block the browser from updating the screen. It makes the app more responsive. The most common example of effects which don't require a cleanup are manual DOM mutations, Network requests, Logging, etc.

Effects with Cleanup

Some effects require cleanup after DOM updation. For example, if we want to set up a subscription to some external data source, it is important to clean up memory so that we don't introduce a memory leak. React performs the cleanup of memory when the component unmounts. However, as we know that, effects run for every render method and not just once. Therefore, React also cleans up effects from the previous render before running the effects next time.

Custom Hooks

A custom Hook is a JavaScript function. The name of custom Hook starts with "use" which can call other Hooks. A custom Hook is just like a regular function, and the word "use" in the beginning tells that this function follows the rules of Hooks. Building custom Hooks allows you to extract component logic into reusable functions.

Let us understand how custom Hooks works in the following example.

```
import React, { useState, useEffect } from 'react';

const useDocumentTitle = title => {
  useEffect(() => {
    document.title = title;
  }, [title])
}

function CustomCounter() {
  const [count, setCount] = useState(0);
  const incrementCount = () => setCount(count + 1);
  useDocumentTitle(`You clicked ${count} times`);
  // useEffect(() => {
  //   document.title = `You clicked ${count} times`
  // });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>Click me</button>
    </div>
  )
}
export default CustomCounter;
```

In the above snippet, useDocumentTitle is a custom Hook which takes an argument as a string of text which is a title. Inside this Hook, we call useEffect Hook and set the title as long as the title has changed. The second argument will perform that check and update the title only when its local state is different than what we are passing in.

Note: A custom Hook is a convention which naturally follows from the design of Hooks, instead of a React feature.

Built-in Hooks

Here, we describe the APIs for the built-in Hooks in React. The built-in Hooks can be divided into two parts, which are given below.

Basic Hooks

- useState
- useEffect
- useContext

Additional Hooks

- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

unit-38

React Flux Concept

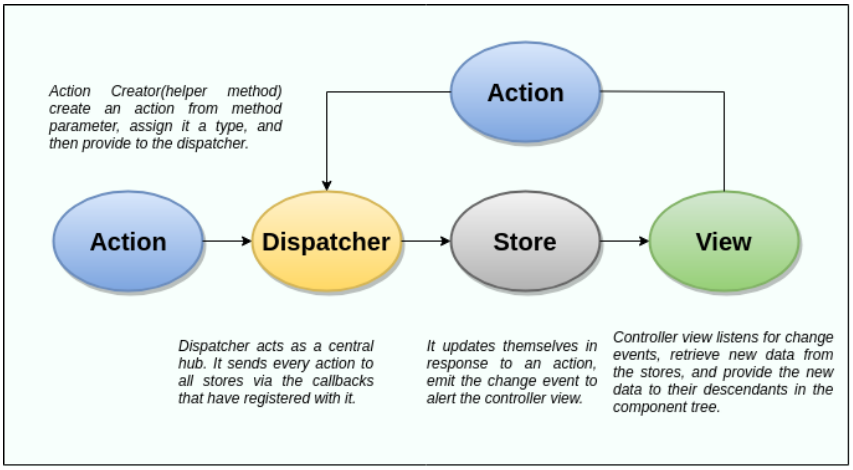
Flux is an application architecture that Facebook uses internally for building the client-side web application with React. It is not a library nor a framework. It is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model. It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner. It reduces the runtime errors.

Flux applications have three major roles in dealing with data:

1. Dispatcher
2. Stores
3. Views (React components)

Here, you should not be confused with the Model-View-Controller (MVC) model. Although, Controllers exists in both, but Flux controller-views (views) found at the top of the hierarchy. It retrieves data from the stores and then passes this data down to their children. Additionally, action creators - dispatcher helper methods used to describe all changes that are possible in the application. It can be useful as a fourth part of the Flux update cycle.

Structure and Data Flow


Action Creator(helper method) create an action from method parameter, assign it a type, and then provide to the dispatcher.
Action
Action
Dispatcher
Store
View
Dispatcher acts as a central hub. It sends every action to all stores via the callbacks that have registered with it.
It updates themselves in response to an action, emit the change event to alert the controller view.
Controller view listens for change events, retrieve new data from the stores, and provide the new data to their descendants in the component tree.

In Flux application, data flows in a single direction(unidirectional). This data flow is central to the flux pattern. The dispatcher, stores, and views are independent nodes with inputs and outputs. The actions are simple objects that contain new data and type property. Now, let us look at the various components of flux architecture one by one.

Dispatcher

It is a central hub for the React Flux application and manages all data flow of your Flux application. It is a registry of callbacks into the stores. It has no real intelligence of its own, and simply acts as a mechanism for distributing the actions to the stores. All stores register itself and provide a callback. It is a place which handled all events that modify the store. When an action creator provides a new action to the dispatcher, all stores receive that action via the callbacks in the registry.

The dispatcher's API has five methods. These are:

SN	Methods	Descriptions
1.	register()	It is used to register a store's action handler callback.
2.	unregister()	It is used to unregisters a store's callback.
3.	waitFor()	It is used to wait for the specified callback to run first.
4.	dispatch()	It is used to dispatches an action.
5.	isDispatching()	It is used to checks if the dispatcher is currently dispatching an action.

Stores

It primarily contains the application state and logic. It is similar to the model in a traditional MVC. It is used for maintaining a particular state within the application, updates themselves in response to an action, and emit the change event to alert the controller view.

Views

It is also called as controller-views. It is located at the top of the chain to store the logic to generate actions and receive new data from the store. It is a React component listen to change events and receives the data from the stores and re-render the application.

Actions

The dispatcher method allows us to trigger a dispatch to the store and include a payload of data, which we call an action. It is an action creator or helper methods that pass the data to the dispatcher.

Advantage of Flux

- It is a unidirectional data flow model which is easy to understand.
- It is open source and more of a design pattern than a formal framework like MVC architecture.
- The flux application is easier to maintain.
- The flux application parts are decoupled.

Unit-39

React Flux Vs. MVC

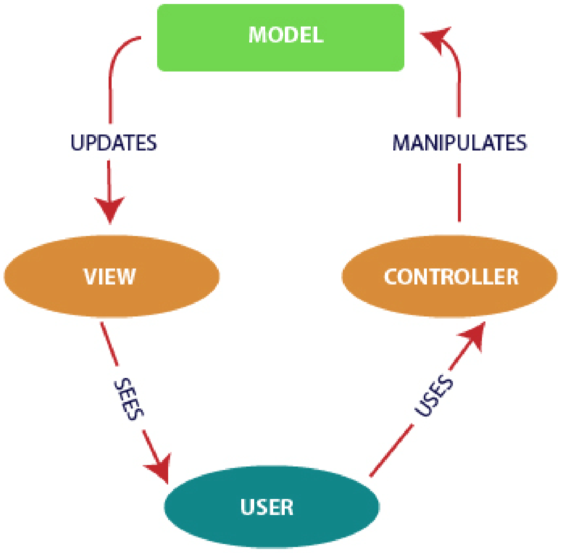
MVC

MVC stands for Model View Controller. It is an architectural pattern used for developing the user interface. It divides the application into three different logical components: The Model, the View, and the Controller. It is first introduced in 1976 in the Smalltalk programming language. In MVC, each component is built to handle specific development aspect of an application. It is one of the most used web development frameworks to create scalable projects.

MVC Architecture

The MVC architecture contains the three components. These are:

- **Model:** It is responsible for maintaining the behavior and data of an application.
- **View:** It is used to display the model in the user interface.
- **Controller:** It acts as an interface between the Model and the View components. It takes user input, manipulates the data(model) and causes the view to update.



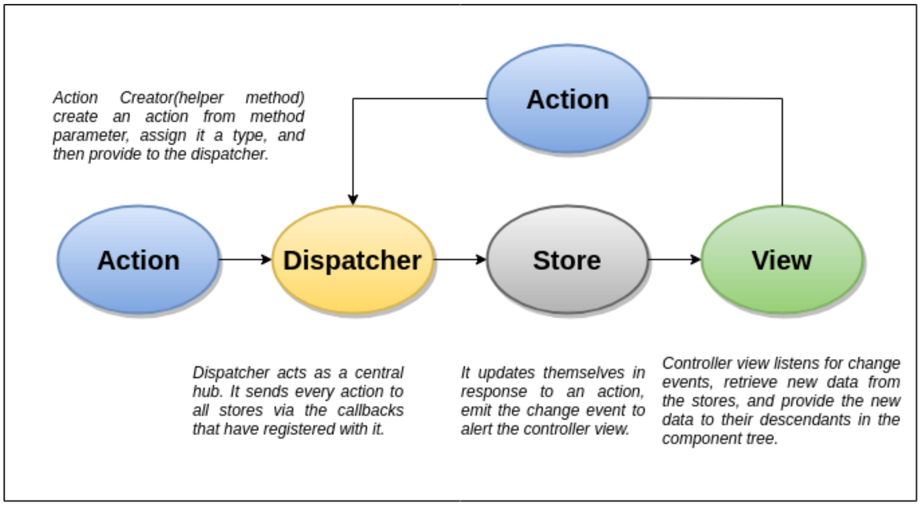
MVC Architecture

Flux

According to the official site, Flux is the application architecture that Facebook uses for building client-side web applications. It is an alternative to MVC architecture and other software design patterns for managing how data flows in the react application. It is the backbone of all React application. It is not a library nor a framework. It complements React as view and follows the concept of Unidirectional Data Flow model.

Flux Architecture has three major roles in dealing with data:

1. Dispatcher
2. Stores
3. Views (React components)



Action Creator(helper method) create an action from method parameter, assign it a type, and then provide to the dispatcher.

Dispatcher acts as a central hub. It sends every action to all stores via the callbacks that have registered with it.

It updates themselves in response to an action, emit the change event to alert the controller view.

Controller view listens for change events, retrieve new data from the stores, and provide the new data to their descendants in the component tree.

MVC Vs. Flux

SN	MVC	FLUX
1.	It was introduced in 1976.	It was introduced just a few years ago.
2.	It supports Bi-directional data Flow model.	It supports Uni-directional data flow model.

3.	In this, data binding is the key.	In this, events or actions are the keys.
4.	It is synchronous.	It is asynchronous.
5.	Here, controllers handle everything(logic).	Here, stores handle all logic.
6.	It is hard to debug.	It is easy to debug because it has common initiating point: Dispatcher.
7.	It is difficult to understand as the project size increases.	It is easy to understand.
8.	Its maintainability is difficult as the project scope goes huge.	Its maintainability is easy and reduces runtime errors.
9.	Testing of application is difficult.	Testing of application is easy.
10.	Scalability is complex.	It can be easily scalable.

Unit-40

React Redux

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by **Dan Abramov** and **Andrew Clark** in **2015**.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch **Actions** to the **Store** to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

Redux was inspired by Flux. Redux studied the Flux architecture and omitted unnecessary complexity.

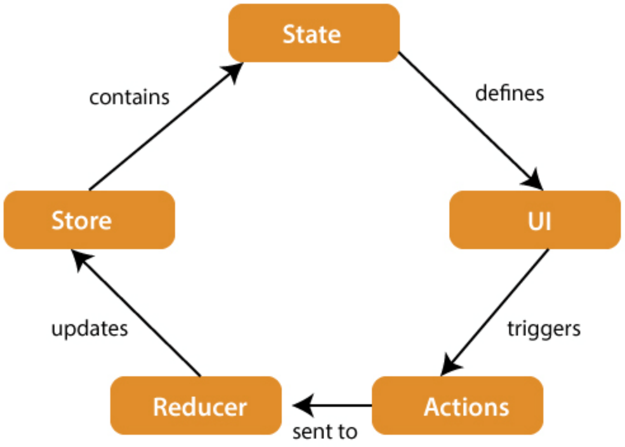
- Redux does not have Dispatcher concept.
- Redux has an only Store whereas Flux has many Stores.
- The Action objects will be received and handled directly by Store.

Why use React Redux?

The main reason to use React Redux are:

- React Redux is the official **UI bindings** for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.
- It encourages good 'React' architecture.
- It implements many performance optimizations internally, which allows to components re-render only when it actually needs.

Redux Architecture



The components of Redux architecture are explained below.

STORE: A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

ACTION: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

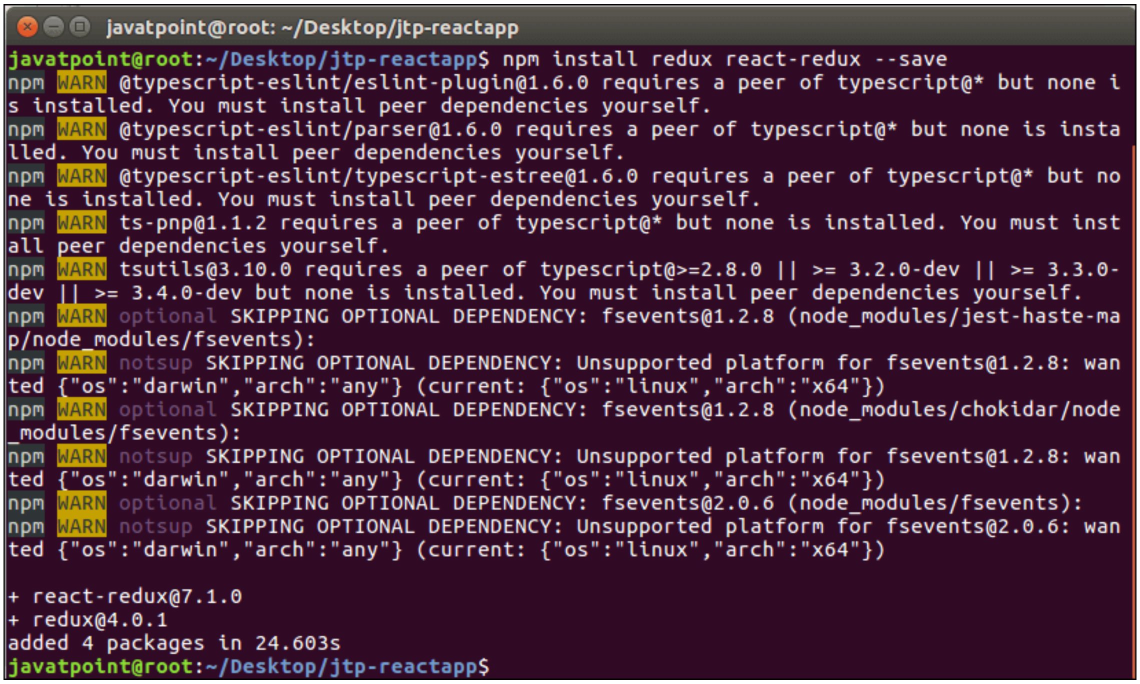
REDUCER: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

Redux Installation

Requirements: React Redux requires React 16.8.3 or later version.

To use React Redux with React application, you need to install the below command.

1. $ npm install redux react-redux --save



```
javatpoint@root: ~/Desktop/jtp-reactapp
javatpoint@root:~/Desktop/jtp-reactapp$ npm install redux react-redux --save
npm WARN @typescript-eslint/eslint-plugin@1.6.0 requires a peer of typescript@* but none i
s installed. You must install peer dependencies yourself.
npm WARN @typescript-eslint/parser@1.6.0 requires a peer of typescript@* but none is insta
lled. You must install peer dependencies yourself.
npm WARN @typescript-eslint/typescript-estree@1.6.0 requires a peer of typescript@* but no
ne is installed. You must install peer dependencies yourself.
npm WARN ts-pnp@1.1.2 requires a peer of typescript@* but none is installed. You must inst
all peer dependencies yourself.
npm WARN tsutils@3.10.0 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >= 3.3.0-
dev || >= 3.4.0-dev but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.8 (node_modules/jest-haste-ma
p/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.8: wan
ted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.8 (node_modules/chokidar/node
_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.8: wan
ted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.0.6 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.0.6: wan
ted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ react-redux@7.1.0
+ redux@4.0.1
added 4 packages in 24.603s
javatpoint@root:~/Desktop/jtp-reactapp$
```

Unit-41

React Redux Example

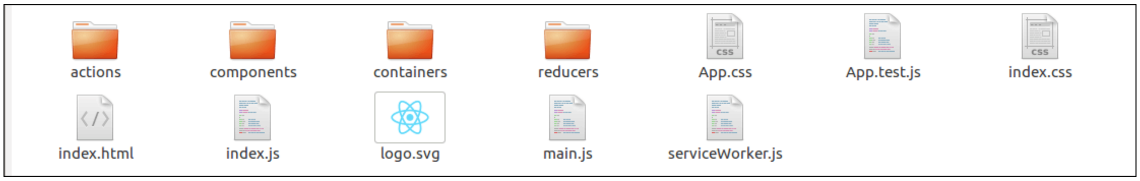
In this section, we will learn how to implements Redux in React application. Here, we provide a simple example to connect Redux and React.

Step-1 Create a new react project using **create-react-app** command. I choose the project name: "**reactproject**." Now, install **Redux** and **React-Redux**.

```
javatpoint@root:~/Desktop$ npx create-react-app reactproject
javatpoint@root:~/Desktop/reactproject$ npm install redux react-redux --save
```

Step-2 Create Files and Folders

In this step, we need to create folders and files for actions, reducers, components, and containers. After creating folders and files, our project looks like as below image.



Step-3 Actions

It uses '**type**' property to inform about data that should be sent to the **Store**. In this folder, we will create two files: **index.js** and **index.spec.js**. Here, we have created an **action creator** that returns our action and sets an **id** for every created item.

Index.js

```
let nextTodoId = 0
export const addTodo = text => ({
  type: 'ADD_TODO',
  id: nextTodoId++,
  text
})

```

```
export const setVisibilityFilter = filter => ({
  type: 'SET_VISIBILITY_FILTER',
  filter
})

export const toggleTodo = id => ({
  type: 'TOGGLE_TODO',
  id
})

export const VisibilityFilters = {
  SHOW_ALL: 'SHOW_ALL',
  SHOW_COMPLETED: 'SHOW_COMPLETED',
  SHOW_ACTIVE: 'SHOW_ACTIVE'
}
```

Index.spec.js

```
import * as actions from './index'

describe('todo actions', () => {
  it('addTodo should create ADD_TODO action', () => {
    expect(actions.addTodo('Use Redux')).toEqual({
      type: 'ADD_TODO',
      id: 0,
      text: 'Use Redux'
    })
  })

  it('setVisibilityFilter should create SET_VISIBILITY_FILTER action', () => {
    expect(actions.setVisibilityFilter('active')).toEqual({
      type: 'SET_VISIBILITY_FILTER',
      filter: 'active'
    })
  })

  it('toggleTodo should create TOGGLE_TODO action', () => {
    expect(actions.toggleTodo(1)).toEqual({
      type: 'TOGGLE_TODO',
      id: 1
    })
  })
})
```

Step-4 Reducers

As we know, Actions only trigger changes in the app, and the Reducers specify those changes. The Reducer is a function which takes two parameters 'Action' and 'State' to calculate and return an updated State. It read the payloads from the 'Actions' and then updates the 'Store' via the State accordingly.

In the given files, each Reducer managing its own part of the global State. The State parameter is different for every Reducer and corresponds to the part of the 'State' it manages. When the app becomes larger, we can split the Reducers into separate files and keep them completely independent and managing different data domains.

Here, we are using 'combineReducers' helper function to add any new Reducers we might use in the future.

index.js

```
import { combineReducers } from 'redux'
import todos from './todos'
import visibilityFilter from './visibilityFilter'

export default combineReducers({
  todos,
  visibilityFilter
})
```

Todos.js

```
const todos = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          id: action.id,
          text: action.text,
          completed: false
        }
      ]
    case 'TOGGLE_TODO':
      return state.map(todo =>
        (todo.id === action.id)
          ? {...todo, completed: !todo.completed}
          : todo
      )
    default:
      return state
  }
}
export default todos
```

Todos.spec.js

```
import todos from './todos'

describe('todos reducer', () => {
  it('should handle initial state', () => {
    expect(
      todos(undefined, {})
    ).toEqual([])
  })

  it('should handle ADD_TODO', () => {
    expect(
      todos([], {
        type: 'ADD_TODO',
        text: 'Run the tests',
        id: 0
      })
    ).toEqual([
      {
        text: 'Run the tests',
        completed: false,
        id: 0
      }
    ])

    expect(
      todos([
        {
          text: 'Run the tests',
          completed: false,
          id: 0
        }
      ], {
        type: 'ADD_TODO',
        text: 'Use Redux',
        id: 1
      })
    ).toEqual([
      {
        text: 'Run the tests',
        completed: false,
        id: 0
      }, {
        text: 'Use Redux',
        completed: false,
        id: 1
      }
    ])

    expect(
      todos([
        {
          text: 'Run the tests',
          completed: false,
          id: 0
        }, {
          text: 'Use Redux',
          completed: false,
          id: 1
        }
      ], {
        type: 'ADD_TODO',
        text: 'Fix the tests',
        id: 2
      })
    ).toEqual([
      {
        text: 'Run the tests',
        completed: false,
        id: 0
      }, {
        text: 'Use Redux',
        completed: false,
        id: 1
      }, {
        text: 'Fix the tests',
        completed: false,
        id: 2
      }
    ])
  })

  it('should handle TOGGLE_TODO', () => {
    expect(
      todos([
        {
          text: 'Run the tests',
          completed: false,
          id: 1
        }, {
          text: 'Use Redux',
          completed: false,
          id: 0
        }
      ], {
        type: 'TOGGLE_TODO',
        id: 1
      })
    ).toEqual([
      {
          text: 'Run the tests',
          completed: true,
          id: 1
        }, {
          text: 'Use Redux',
          completed: false,
          id: 0
        }
      ])
    })
          })
```

VisibilityFilter.js

```
import { VisibilityFilters } from '../actions'

const visibilityFilter = (state = VisibilityFilters.SHOW_ALL, action) => {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return action.filter
    default:
      return state
  }
}
export default visibilityFilter
```

Step-5 Components

It is a Presentational Component, which concerned with how things look such as markup, styles. It receives data and invokes callbacks exclusively via props. It does not know where the data comes from or how to change it. It only renders what is given to them.

App.js

It is the root component which renders everything in the UI.

```
import React from 'react'
import Footer from './Footer'
import AddTodo from '../containers/AddTodo'
import VisibleTodoList from '../containers/VisibleTodoList'

const App = () => (
  <div>
    <AddTodo />
    <VisibleTodoList />
    <Footer />
  </div>
)
export default App
```

Footer.js

It tells where the user changes currently visible **todos**.

```
import React from 'react'
import FilterLink from '../containers/FilterLink'
import { VisibilityFilters } from '../actions'

const Footer = () => (
  <p>
    Show: <FilterLink filter={VisibilityFilters.SHOW_ALL}>All</FilterLink>
    {', '}
    <FilterLink filter={VisibilityFilters.SHOW_ACTIVE}>Active</FilterLink>
    {', '}
    <FilterLink filter={VisibilityFilters.SHOW_COMPLETED}>Completed</FilterLink>
  </p>
)
export default Footer
```

Link.js

It is a link with a callback.

```
import React from 'react'
import PropTypes from 'prop-types'

const Link = ({ active, children, onClick }) => {
  if (active) {
    return <span>{children}</span>
  }

  return (
    <a
      href=""
      onClick={e => {
        e.preventDefault()
        onClick()
      }}
    >
      {children}
    </a>
  )
}

Link.propTypes = {
  active: PropTypes.bool.isRequired,
  children: PropTypes.node.isRequired,
  onClick: PropTypes.func.isRequired
}

export default Link
```

Todo.js

It represents a single todo item which shows **text**.

```
import React from 'react'
import PropTypes from 'prop-types'

const Todo = ({ onClick, completed, text }) => (
  <li
    onClick={onClick}
    style={{
      textDecoration: completed ? 'line-through' : 'none'
    }}
  >
    {text}
  </li>
)

Todo.propTypes = {
  onClick: PropTypes.func.isRequired,
  completed: PropTypes.bool.isRequired,
  text: PropTypes.string.isRequired
}

export default Todo
```

TodoList.js

It is a list to show visible todos{ id, text, completed }.

```
import React from 'react'
import PropTypes from 'prop-types'
import Todo from './Todo'

const TodoList = ({ todos, onTodoClick }) => (
  <ul>
    {todos.map((todo, index) => (
      <Todo key={index} {...todo} onClick={() => onTodoClick(index)} />
    ))}
  </ul>
)

TodoList.propTypes = {
  todos: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      completed: PropTypes.bool.isRequired,
      text: PropTypes.string.isRequired
    }).isRequired
  ).isRequired,
  onTodoClick: PropTypes.func.isRequired
}
export default TodoList
```

Step-6 Containers

It is a Container Component which concerned with how things work such as data fetching, updates State. It provides data and behavior to presentational components or other container components. It uses Redux State to read data and dispatch Redux Action for updating data.

AddTodo.js

It contains the input field with an ADD (submit) button.

```
import React from 'react'
import { connect } from 'react-redux'
import { addTodo } from '../actions'

const AddTodo = ({ dispatch }) => {
  let input

  return (
    <div>
      <form onSubmit={e => {
        e.preventDefault()
        if (!input.value.trim()) {
          return
        }
        dispatch(addTodo(input.value))
        input.value = ''
      }}>
        <input ref={node => input = node} />
        <button type="submit">
          Add Todo
        </button>
      </form>
    </div>
  )
}
export default connect()(AddTodo)
```

FilterLink.js

It represents the current visibility filter and renders a link.

```
import { connect } from 'react-redux'
import { setVisibilityFilter } from '../actions'
import Link from '../components/Link'

const mapStateToProps = (state, ownProps) => ({
  active: ownProps.filter === state.visibilityFilter
})

const mapDispatchToProps = (dispatch, ownProps) => ({
  onClick: () => dispatch(setVisibilityFilter(ownProps.filter))
})

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Link)
```

VisibleTodoList.js

It filters the todos and renders a TodoList.

```
import { connect } from 'react-redux'
import { toggleTodo } from '../actions'
import TodoList from '../components/TodoList'
import { VisibilityFilters } from '../actions'

const getVisibleTodos = (todos, filter) => {
  switch (filter) {
    case VisibilityFilters.SHOW_ALL:
      return todos
    case VisibilityFilters.SHOW_COMPLETED:
      return todos.filter(t => t.completed)
    case VisibilityFilters.SHOW_ACTIVE:
      return todos.filter(t => !t.completed)
    default:
      throw new Error('Unknown filter: ' + filter)
  }
}

const mapStateToProps = state => ({
  todos: getVisibleTodos(state.todos, state.visibilityFilter)
})

const mapDispatchToProps = dispatch => ({
  toggleTodo: id => dispatch(toggleTodo(id))
})
```

```

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(TodoList)
```

Step-7 Store

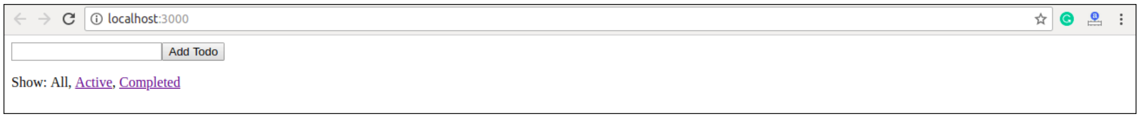
All container components need access to the Redux Store to subscribe to it. For this, we need to pass it(store) as a prop to every container component. However, it gets tedious. So we recommend using special React Redux component called which make the store available to all container components without passing it explicitly. It used once when you render the root component.

index.js

```
import React from 'react'
import { render } from 'react-dom'
import { createStore } from 'redux'
import { Provider } from 'react-redux'
import App from './components/App'
import rootReducer from './reducers'

const store = createStore(rootReducer)

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

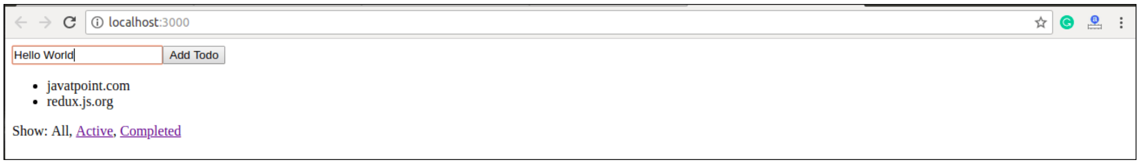
Output

When we execute the application, it gives the output as below screen.



Show: All, Active, Completed

Now, we will be able to add items in the list.



Hello World

- javatpoint.com
- redux.js.org

Show: All, Active, Completed

The detailed explanation of React-Redux example can be shown here: https://redux.js.org/basics/usage-with-react.

Unit-42

React Portals

The **React 16.0** version introduced React portals in **September 2017**. A React portal provides a way to render an element outside of its component hierarchy, i.e., in a separate component.

Before React 16.0 version, it is very tricky to render the child component outside of its parent component hierarchy. If we do this, it breaks the convention where a component needs to render as a new element and follow a **parent-child** hierarchy. In React, the parent component always wants to go where its child component goes. That's why React portal concept comes in.

Syntax

```
ReactDOM.createPortal(child, container)
```

Here, the first argument (child) is the component, which can be an element, string, or fragment, and the second argument (container) is a DOM element.

Example before React v16

Generally, when you want to return an element from a component's render method, it is mounted as a new div into the DOM and render the children of the closest parent component.

```
render() {
// React mounts a new div into the DOM and renders the children into it
  return (
    <div>
```

```
      {this.props.children}
    </div>
  );
}
```

Example using portal

But, sometimes we want to insert a child component into a different location in the DOM. It means React does not want to create a new div. We can do this by creating React portal.

```
render() {
 return ReactDOM.createPortal(
   this.props.children,
   myNode,
 );
}
```

Features

- It uses React version 16 and its official API for creating portals.
- It has a fallback for React version 15.
- It transports its children component into a new React portal which is appended by default to document.body.
- It can also target user specified DOM element.
- It supports server-side rendering
- It supports returning arrays (no wrapper div's needed)
- It uses <Portal /> and <PortalWithState /> so there is no compromise between flexibility and convenience.
- It doesn't produce any DOM mess.
- It has no dependencies, minimalistic.

When to use?

The common use-cases of React portal include:

- Modals
- Tooltips
- Floating menus

- Widgets

Installation

We can install React portal using the following command.

```
$ npm install react-portal --save
```

Explanation of React Portal

Create a new React project using the following command.

```
$ npx create-react-app reactapp
```

Open the App.js file and insert the following code snippet.

App.js

```
import React, {Component} from 'react';
import './App.css'
import PortalDemo from './PortalDemo.js';

class App extends Component {
    render () {
        return (
            <div className='App'>
          <PortalDemo />
    </div>
        );
    }
}
export default App;
```

The next step is to create a **portal** component and import it in the App.js file.

PortalDemo.js

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

function PortalDemo(){
    return ReactDOM.createPortal(
      <h1>Portals Demo</h1>,
      document.getElementById('portal-root')
    )
}
```

```
export default PortalDemo
```

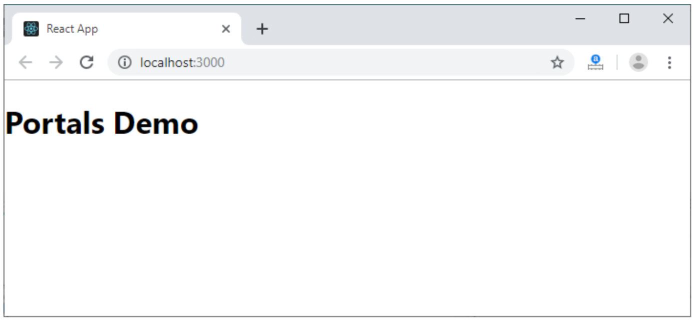
Now, open the Index.html file and add a <div id="portal-root"></div> element to access the child component outside the root node.

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>It is required to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <div id="portal-root"></div>
  </body>
</html>
```

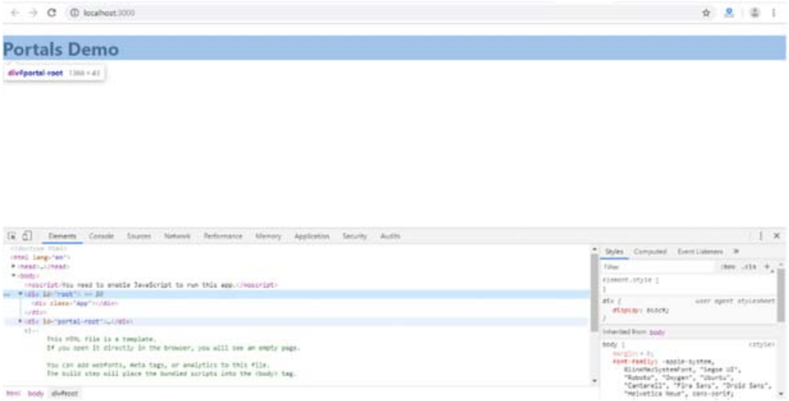
Output:

When we execute the React app, we will get the following screen.



Portals Demo

Now, open the **Inspect** (ctrl + shift + I). In this window, select the **Elements** section and then click on the <div id="portal-root"></div> component. Here, we can see that each tag is under the "portal-root" DOM node, not the "root" DOM node. Hence, we can see that how React Portal provides the ability to break out of root DOM tree.



Portals Demo

```
<!doctype html>
<html lang="en">
▶ <head>…</head>
▼ <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
 ▼ <div id="root"> == $0
     <div class="App"></div>
   </div>
 ▶ <div id="portal-root">…</div>
   <!--
        This HTML file is a template.
        If you open it directly in the browser, you will see an empty page.

        You can add webfonts, meta tags, or analytics to this file.
        The build step will place the bundled scripts into the <body> tag.
```

```
element.style {
}
div {                    user agent stylesheet
  display: block;
}
Inherited from body
body {                                  <style>
  margin: 0;
  font-family: -apple-system,
    BlinkMacSystemFont, "Segoe UI",
    "Roboto", "Oxygen", "Ubuntu",
    "Cantarell", "Fira Sans", "Droid Sans",
    "Helvetica Neue", sans-serif;
```

Unit-43

React Error Boundaries

In the past, if we get any JavaScript errors inside components, it corrupts the Reacts internal state and put React in a broken state on next renders. There are no ways to handle these errors in React components, nor it provides any methods to recover from them. But, **React 16** introduces a new concept to handle the errors by using the **error boundaries**. Now, if any JavaScript error found in a part of the UI, it does not break the whole app.

Error boundaries are React components which catch JavaScript errors anywhere in our app, log those errors, and display a fallback UI. It does not break the whole app component tree and only renders the fallback UI whenever an error occurred in a component. Error boundaries catch errors during rendering in component lifecycle methods, and constructors of the whole tree below them.

Note:

Sometimes, it is not possible to catch Error boundaries in React application. These are:

- Event handlers
- Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)
- Server-side rendering
- Errors are thrown in the error boundary itself rather than its children.

For simple React app, we can declare an error boundary once and can use it for the whole application. For a complex application which have multiple components, we can declare multiple error boundaries to recover each part of the entire application.

We can also report the error to an error monitoring service like **Rollbar**. This monitoring service provides the ability to track how many users are affected by errors, find causes of them, and improve the user experience.

Error boundary in class

A class component can becomes an error boundary if it defines a new lifecycle methods either static getDerivedStateFromError() or componentDidCatch(error, info). We can use static getDerivedStateFromError() to render a fallback UI when an error has been thrown, and can use componentDidCatch() to log error information.

An error boundary can?t catch the error within itself. If the error boundary fails to render the error message, the error will go to the closest error boundary above it. It is similar to the catch {} block in JavaScript.

How to implement error boundaries

Step-1 Create a class which extends React component and passes the props inside it.

Step-2 Now, add componentDidCatch() method which allows you to catch error in the components below them in the tree.

Step-3 Next add render() method, which is responsible for how the component should be rendered. For example, it will display the error message like "Something is wrong."

Example

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
  static getDerivedStateFromError(error) {
    // It will update the state so the next render shows the fallback UI.
    return { hasError: true };
  }
  componentDidCatch(error, info) {
    // It will catch error in any component below. We can also log the error to an error reporting service.
    logErrorToMyService(error, info);
  }
  render() {
    if (this.state.hasError) {
        return (
        <div>Something is wrong.</div>;
    );
    }
    return this.props.children;
  }
}
```

Step-4 Now, we can use it as a regular component. Add the new component in HTML, which you want to include in the error boundary. In this example, we are adding an error boundary around a MyWidgetCounter component.

```
<ErrorBoundary>
       <MyWidgetCounter/>
</ErrorBoundary>
```

Where to Place Error Boundaries

An error boundary entirely depends on you. You can use error boundaries on the top-level of the app components or wrap it on the individual components to protect them from breaking the other parts of the app.

Let us see an example.

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

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { error: false, errorInfo: null };
  }

  componentDidCatch(error, errorInfo) {
    // Catch errors in any components below and re-render with error message
    this.setState({
      error: error,
      errorInfo: errorInfo
    })
  }

  render() {
    if (this.state.errorInfo) {
      return (
        <div>
          <h2>Something went wrong.</h2>
          <details style={{ whiteSpace: 'pre-wrap' }}>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo.componentStack}
          </details>
        </div>
      );
    }
    return this.props.children;
  }
}

class BuggyCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { counter: 0 };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(({counter}) => ({
      counter: counter + 1
    }));
  }

  render() {
    if (this.state.counter === 3) {
      throw new Error('I crashed!');
    }
    return <h1 onClick={this.handleClick}>{this.state.counter}</h1>;
  }
}

function App() {
  return (
    <div>
      <p><b>Example of Error Boundaries</b></p>
      <hr />
      <ErrorBoundary>
        <p>These two counters are inside the same error boundary.</p>
          <BuggyCounter />
          <BuggyCounter />
      </ErrorBoundary>
      <hr />
      <p>These two counters are inside of their individual error boundary.</p>
        <ErrorBoundary><BuggyCounter /></ErrorBoundary>
        <ErrorBoundary><BuggyCounter /></ErrorBoundary>
    </div>
  );
}
export default App
```

In the above code snippet, when we click on the **numbers**, it increases the **counters**. The counter is programmed to **throw** an error when it reaches **3**. It simulates a JavaScript error in a component. Here, we used an error boundary in **two ways**, which are given below.

First, these two counters are inside the same error boundary. If anyone crashes, the error boundary will replace both of them.

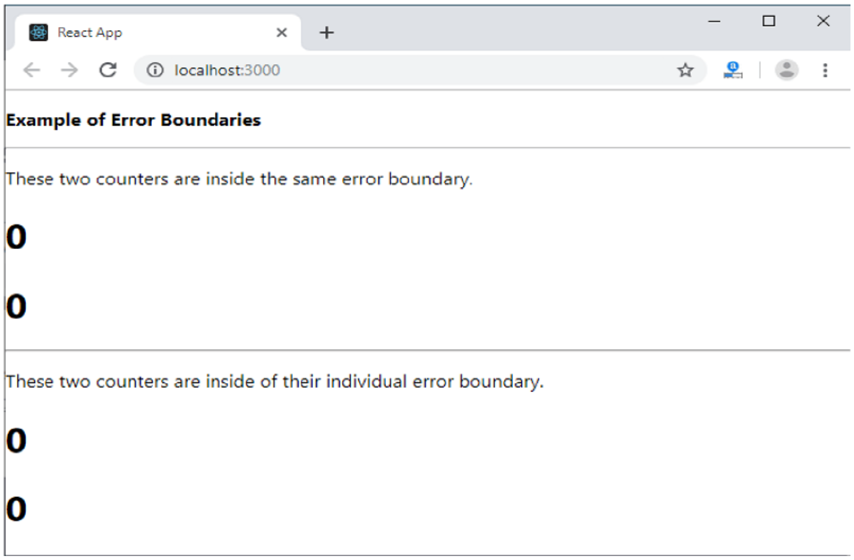
```
<ErrorBoundary>
        <BuggyCounter />
        <BuggyCounter />
</ErrorBoundary>
```

Second, these two counters are inside of their individual error boundary. So if anyone crashes, the other is not affected.

```
<ErrorBoundary><BuggyCounter /></ErrorBoundary>
<ErrorBoundary><BuggyCounter /></ErrorBoundary>
```

Output:

When we execute the above code, we will get the following output.



Example of Error Boundaries

These two counters are inside the same error boundary.

0

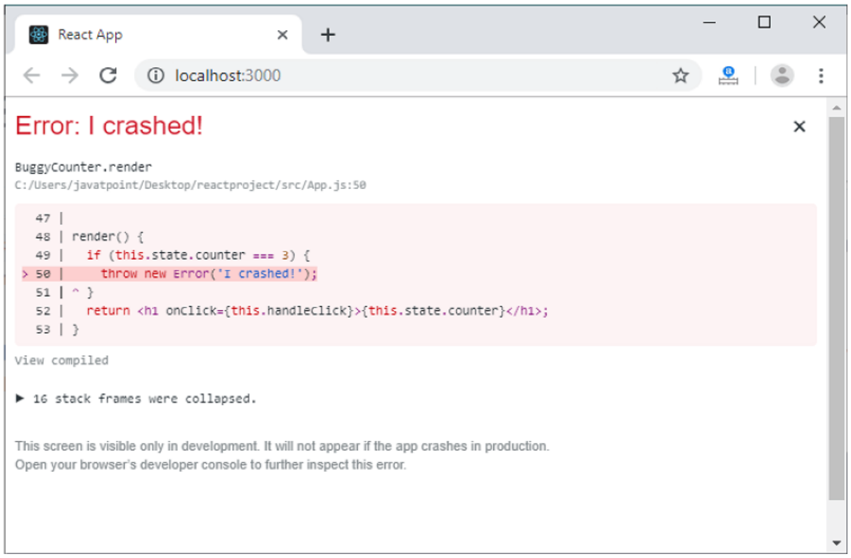
0

These two counters are inside of their individual error boundary.

0

0

When the counter has reached at 3, it gives the following output.



```
Error: I crashed!

BuggyCounter.render
C:/Users/javatpoint/Desktop/reactproject/src/App.js:50

  47 |
  48 | render() {
  49 |   if (this.state.counter === 3) {
> 50 |     throw new Error('I crashed!');
  51 | ^ }
  52 |   return <h1 onClick={this.handleClick}>{this.state.counter}</h1>;
  53 | }

View compiled

▶ 16 stack frames were collapsed.

This screen is visible only in development. It will not appear if the app crashes in production.
Open your browser's developer console to further inspect this error.
```

New Behavior for Uncaught error

It is an important implication related to error boundaries. If the error does not catch by any error boundary, it will result in **unmounting** of the whole React application.

Error Boundary in Event Handler

Error boundaries do not allow catching errors inside event handlers. React does not need any error boundary to recover from errors in the event handler. If there is a need to catch errors in the event handler, you can use JavaScript **try-catch** statement.

In the below example, you can see how an event handler will handle the errors.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { error: null };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    try {
      // Do something which can throw error
    } catch (error) {
      this.setState({ error });
    }
  }

  render() {
    if (this.state.error) {
      return
          <h2>It caught an error.</h2>
    }
    return <div onClick={this.handleClick}>Click Me</div>
  }
}
```

43. React Interview Questions and Answers

A list of top frequently asked **React Interview Questions and Answers** are given below.

SN	React Interview Topic
1	General React Interview Questions
2	React Component Interview Questions
3	React Refs Interview Questions
4	React Router Interview Questions
5	React Styling Interview Questions
6	React Redux Interview Questions

General React Interview Questions

1) What is React?

React is a declarative, efficient, flexible open source front-end JavaScript library developed by Facebook in 2011. It follows the component-based approach for building reusable UI components, especially for single page application. It is used for developing interactive view layer of web and mobile apps. It was created by Jordan Walke, a software engineer at Facebook. It was initially deployed on Facebook's News Feed section in 2011 and later used in its products like WhatsApp & Instagram.

For More Information, ***Click here***.

2) What are the features of React?

React framework gaining quick popularity as the best framework among web developers. The main features of React are:

- JSX
- Components
- One-way Data Binding
- Virtual DOM
- Simplicity
- Performance

For More Information, ***Click here***.

3) What are the advantages of React?

The advantages of React are:

- Easy to Learn and USe
- Creating Dynamic Web Applications Becomes Easier
- Reusable Components
- Performance Enhancement
- The Support of Handy Tools
- Known to be SEO Friendly
- The Benefit of Having JavaScript Library
- Scope for Testing the Codes

For More Information, ***Click here***.

4) What are the limitations of React?

The limitations of React are:

- The high pace of development

- Poor Documentation
- View Part
- JSX as a barrier

For More Information, ***Click here***.

5) What is JSX?

JSX stands for JavaScript XML. It is a React extension which allows writing JavaScript code that looks similar to HTML. It makes HTML file easy to understand. The JSX file makes the React application robust and boosts its performance. JSX provides you to write XML-like syntax in the same file where you write JavaScript code, and then preprocessor (i.e., transpilers like Babel) transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

Example

```
class App extends React.Component {
  render() {
    return(
      <div>
        <h1>Hello JavaTpoint</h1>
      </div>
    )
  }
}
```

In the above example, text inside <h1> tag return as JavaScript function to the render function. After compilation, the JSX expression becomes a normal JavaScript function, as shown below.

```
React.createElement("h1", null, "Hello JavaTpoint");
```

For More Information, ***Click here***.

6) Why can't browsers read JSX?

Browsers cannot read JSX directly because they can only understand JavaScript objects, and JSX is not a regular JavaScript object. Thus, we need to transform the JSX file into a JavaScript object using transpilers like Babel and then pass it to the browser.

7) Why we 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.
- t is type-safe, and most of the errors can be found at compilation time.
- It makes easier to create templates.

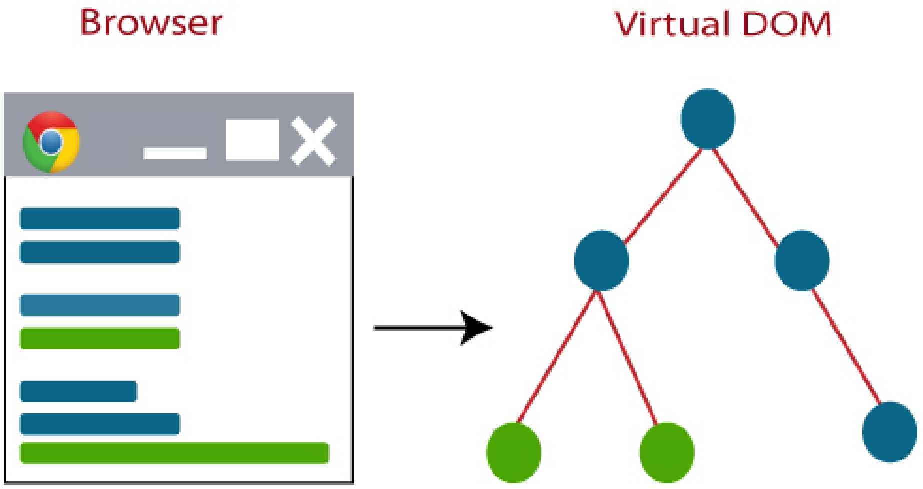
8) What do you understand by Virtual DOM?

A Virtual DOM is a lightweight JavaScript object which is an in-memory representation of real DOM. It is an intermediary step between the render function being called and the displaying of elements on the screen. It is similar to a node tree which lists the elements, their attributes, and content as objects and their properties. The render function creates a node tree of the React components and then updates this node tree in response to the mutations in the data model caused by various actions done by the user or by the system.

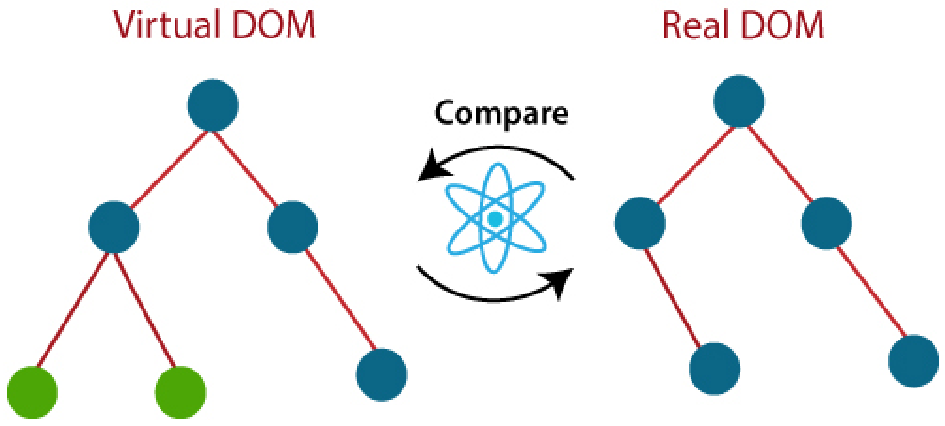
9) Explain the working of Virtual DOM.

Virtual DOM works in three steps:

1. Whenever any data changes in the React App, the entire UI is re-rendered in Virtual DOM representation.



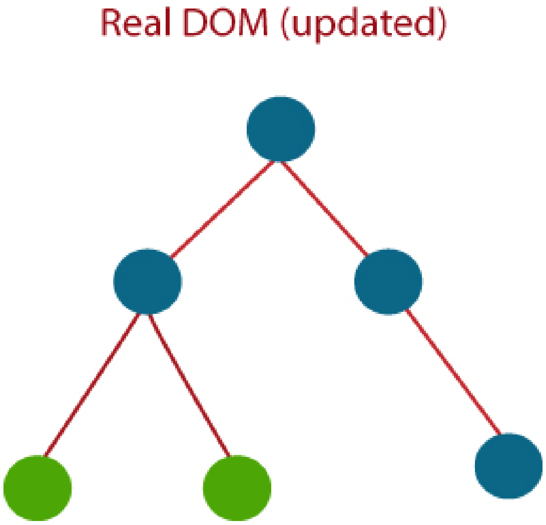
2. Now, the difference between the previous DOM representation and the new DOM is calculated.



3. Once the calculations are completed, the real DOM updated with only those things which are changed.

	Angular	React
Author	Google	Facebook Community
Developer	Misko Hevery	Jordan Walke
Initial Release	October 2010	March 2013
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open Source JS Framework
Rendering	Client-Side	Server-Side
Data-Binding	Bi-directional	Uni-directional
DOM	Regular DOM	Virtual DOM

Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux
Performance	Slow	Fast, due to virtual DOM.



10) How is React different from Angular?

The React is different from Angular in the following ways.

For More Information, ***Click here***.

11) How React's ES6 syntax is different from ES5 syntax?

The React's ES6 syntax has changed from ES5 syntax in the following aspects.

require vs. Import

```
// ES5
var React = require('react');

// ES6
import React from 'react';
```

exports vs. export

```
// ES5
module.exports = Component;

// ES6
export default Component;
```

component and function

```
// ES5
var MyComponent = React.createClass({
    render: function() {
        return(
          <h3>Hello JavaTpoint</h3>
        );
    }
});

// ES6
class MyComponent extends React.Component {
    render() {
        return(
          <h3>Hello Javatpoint</h3>
        );
    }
}
```

props

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string },
    render: function() {
        return(
           <h3>Hello, {this.props.name}!</h3>
        );
    }
});

// ES6
class App extends React.Component {
    render() {
        return(
          <h3>Hello, {this.props.name}!</h3>
        );
    }
}
```

state

```
var App = React.createClass({
    getInitialState: function() {
        return { name: 'world' };
    },
    render: function() {
        return(
          <h3>Hello, {this.state.name}!</h3>
        );
    }
});

// ES6
class App extends React.Component {
    constructor() {
        super();
        this.state = { name: 'world' };
    }
    render() {
        return(
          <h3>Hello, {this.state.name}!</h3>
        );
    }
}
```

12) What is the difference between ReactJS and React Native?

The main differences between ReactJS and React Native are given below.

SN	ReactJS	React Native
1.	Initial release in 2013.	Initial release in 2015.

2.	It is used for developing web applications.	It is used for developing mobile applications.
3.	It can be executed on all platforms.	It is not platform independent. It takes more effort to be executed on all platforms.
4.	It uses a JavaScript library and CSS for animations.	It comes with built-in animation libraries.
5.	It uses React-router for navigating web pages.	It has built-in Navigator library for navigating mobile applications.
6.	It uses HTML tags.	It does not use HTML tags.
7.	In this, the Virtual DOM renders the browser code.	In this, Native uses its API to render code for mobile applications.

React Component Interview Questions

13) What do you understand from "In React, everything is a component."

In React, components are the building blocks of React applications. These components divide the entire React application's UI into small, independent, and reusable pieces of code. React renders each of these components independently without affecting the rest of the application UI. Hence, we can say that, in React, everything is a component.

14) Explain the purpose of render() in React.

It is mandatory for each React component to have a render() function. Render function is used to return the HTML which you want to display in a component. If you need to rendered more than one HTML element, you need to grouped together inside single enclosing tag (parent tag) such as <div>, <form>, <group> etc. This function returns the same result each time it is invoked.

Example: If you need to display a heading, you can do this as below.

```
import React from 'react'

class App extends React.Component {
  render (){
    return (
      <h1>Hello World</h1>
    )
  }
}
export default App
```

Points to Note:

- Each render() function contains a return statement.
- The return statement can have only one parent HTML tag.

15) How can you embed two or more components into one?

You can embed two or more components into the following way:

```
import React from 'react'

class App extends React.Component {
  render (){
    return (
      <h1>Hello World</h1>
    )
  }
}

class Example extends React.Component {
  render (){
    return (
      <h1>Hello JavaTpoint</h1>
    )
  }
}
export default App
```

16) What is Props?

Props stand for "Properties" in React. They are read-only inputs to components. Props are 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 the parent to the child components throughout the application.

It is similar to function arguments and 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.

For More Information, ***Click here***.

17) What is a State in React?

The State is an updatable structure which holds the data and information about the component. It may be changed over the lifetime of the component in response to user action or system event. It is the heart of the react component which determines the behavior of the component and how it will render. It must be kept as simple as possible.

Let's create a "User" component with "message state."

```
import React from 'react'

class User extends React.Component {
  constructor(props) {
    super(props)

    this.state = {
      message: 'Welcome to JavaTpoint'
    }
  }

  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    )
  }
}
export default User
```

For More Information, ***Click here***.

18) Differentiate between States and Props.

The major differences between States and Props are given below.

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	The stateless component can have Props.	The stateless components cannot have State.

7.	Props make components reusable.	The State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the component itself.

19) How can you update the State of a component?

We can update the State of a component using this.setState() method. This method does not always replace the State immediately. Instead, it only adds changes to the original State. It is a primary method which is used to update the user interface(UI) in response to event handlers and server responses.

Example

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';

class App extends React.Component {
  constructor() {
    super();
    this.state = {
       msg: "Welcome to JavaTpoint"
    };
    this.updateSetState = this.updateSetState.bind(this);
  }
  updateSetState() {
     this.setState({
       msg:"Its a best ReactJS tutorial"
     });
  }
  render() {
    return (
      <div>
         <h1>{this.state.msg}</h1>
         <button onClick = {this.updateSetState}>SET STATE</button>
      </div>
    );
  }
}
export default App;
```

For More Information, ***Click here***.

20) Differentiate between stateless and stateful components.

The difference between stateless and stateful components are:

SN	Stateless Component	Stateful Component
1.	The stateless components do not hold or manage state.	The stateful components can hold or manage state.
2.	It does not contain the knowledge of past, current, and possible future state changes.	It can contain the knowledge of past, current, and possible future changes in state.
3.	It is also known as a functional component.	It is also known as a class component.
4.	It is simple and easy to understand.	It is complex as compared to the stateless component.
5.	It does not work with any lifecycle method of React.	It can work with all lifecycle method of React.
6.	The stateless components cannot be reused.	The stateful components can be reused.

21) What is arrow function in React? How is it used?

The Arrow function is the new feature of the ES6 standard. If you need to use arrow functions, it is not necessary to bind any event to 'this.' Here, the scope of 'this' is global and not limited to any calling function. So If you are using Arrow Function, there is no need to bind 'this' inside the constructor. It is also called 'fat arrow '(=>) functions.

```
//General way
render() {
    return(
        <MyInput onChange={this.handleChange.bind(this) } />
    );
}
//With Arrow Function
render() {
    return(
        <MyInput onChange={ (e) => this.handleOnChange(e) } />
    );
}
```

22) What is an event in React?

An event is an action which triggers as a result of the user action or system generated event like a mouse click, loading of a web page, pressing a key, window resizes, etc. In React, the event handling system is very similar to handling events in DOM elements. The React event handling system is known as Synthetic Event, which is a cross-browser wrapper of the browser's native event.

Handling events with React have some syntactical differences, which are:

- React events are named as camelCase instead of lowercase.
- With JSX, a function is passed as the event handler instead of a string.

For More Information, ***Click here***.

23) How do you create an event in React?

We can create an event as follows.

```
class Display extends React.Component({
    show(msgEvent) {
        // code
    },
    render() {
        // Here, we render the div with an onClick prop
        return (
          <div onClick={this.show}>Click Me</div>
        );
    }
});
```

Example

```
import React, { Component } from 'react';
class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         companyName: ''
      };
   }
   changeText(event) {
      this.setState({
         companyName: event.target.value
      });
   }
   render() {
      return (
         <div>
            <h2>Simple Event Example</h2>
            <label htmlFor="name">Enter company name: </label>
            <input type="text" id="companyName" onChange={this.changeText.bind(this)}/>
            <h4>You entered: { this.state.companyName }</h4>
         </div>
      );
   }
}
export default App;
```

For More Information, ***Click here***.

24) What are synthetic events in React?

A synthetic event is an object which acts as a cross-browser wrapper around the browser's native event. It combines the behavior of different browser's native event into one API, including stopPropagation() and preventDefault().

In the given example, e is a Synthetic event.

```
function ActionLink() {
   function handleClick(e) {
      e.preventDefault();
      console.log('You had clicked a Link.');
   }
```

```
    return (
        <a href="#" onClick={handleClick}>
            Click_Me
        </a>
    );
}
```

25) what is the difference between controlled and uncontrolled components?

The difference between controlled and uncontrolled components are:

SN	Controlled	Uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

26) Explain the Lists in React.

Lists are used to display data in an ordered format. In React, Lists can be created in a similar way as we create it in JavaScript. We can traverse the elements of the list using the map() function.

Example

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

```

function NameList(props) {
  const myLists = props.myLists;
  const listItems = myLists.map((myList) =>
    <li>{myList}</li>
  );
  return (
    <div>
        <h2>Rendering Lists inside component</h2>
              <ul>{listItems}</ul>
    </div>
  );
}
const myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
ReactDOM.render(
  <NameList myLists={myLists} />,
  document.getElementById('app')
);
export default App;
```

For More Information, ***Click here***.

27) What is the significance of keys in React?

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time. It increases application performance.

28) How are forms created in React?

Forms allow the users to interact with the application as well as gather information from the users. Forms can perform many tasks such as user authentication, adding user, searching, filtering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

React offers a stateful, reactive approach to build a form. The forms in React are similar to HTML forms. But in React, the state property of the component is only updated via setState(), and a JavaScript function handles their submission. This function has full access to the data which is entered by the user into a form.

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

```

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value: event.target.value});
  }
  handleSubmit(event) {
    alert('You have submitted the input successfully: ' + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <h1>Controlled Form Example</h1>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
export default App;
```

For More Information, ***Click here***.

29) What are the different phases of React component's lifecycle?

The different phases of React component's lifecycle are:

Initial Phase: It is the birth phase of the React lifecycle when the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component.

Mounting Phase: In this phase, the instance of a component is created and added into the DOM.

Updating Phase: It is the next phase of the React lifecycle. In this phase, we get new Props and change State. This phase can potentially update and re-render only when a prop or state change occurs. The main aim of this phase is to ensure that the component is displaying the latest version of itself. This phase repeats again and again.

Unmounting Phase: It is the final phase of the React lifecycle, where the component instance is destroyed and unmounted(removed) from the DOM.

For More Information, ***Click here***.

30) Explain the lifecycle methods of React components in detail.

The important React lifecycle methods are:

- **getInitialState():** It is used to specify the default value of this.state. It is executed before the creation of the component.
- **componentWillMount():** It is executed before a component gets rendered into the DOM.
- **componentDidMount():** It is executed when the component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
- **componentWillReceiveProps():** It is invoked when a component receives new props from the parent class and before another render is called. If you want to update the State in response to prop changes, you should compare this.props and nextProps to perform State transition by using this.setState() method.
- **shouldComponentUpdate():** It is invoked when a component decides any changes/updation to the DOM and returns true or false value based on certain conditions. If this method returns true, the component will update. Otherwise, the component will skip the updating.
- **componentWillUpdate():** It is invoked before rendering takes place in the DOM. Here, you can't change the component State by invoking this.setState() method. It will not be called, if shouldComponentUpdate() returns false.
- **componentDidUpdate():** It is invoked immediately after rendering takes place. In this method, you can put any code inside this which you want to execute once the updating occurs.
- **componentWillUnmount():** It is invoked immediately before a component is destroyed and unmounted permanently. It is used to clear up the memory spaces such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

For More Information, ***Click here***.

31) What are Pure Components?

Pure components introduced in React 15.3 version. The React.Component and React.PureComponent differ in the shouldComponentUpdate() React lifecycle method. This method decides the re-rendering of the component by returning a boolean value (true or false). In React.Component, shouldComponentUpdate() method returns true by default. But in React.PureComponent, it compares the changes in state or props to re-render the component. The pure component enhances the simplicity of the code and performance of the application.

32) What are Higher Order Components(HOC)?

In React, Higher Order Component is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component. In other words, it is a function which accepts another function as an argument. According to the official website, it is not the feature(part) in React API, but a pattern that emerges from React's compositional nature.

For More Information, ***Click here***.

33) What can you do with HOC?

You can do many tasks with HOC, some of them are given below:

- Code Reusability
- Props manipulation
- State manipulation
- Render highjacking

34) What is the difference between Element and Component?

The main differences between Elements and Components are:

SN	Element	Component
1.	An element is a plain JavaScript object which describes the component state and	A component is the core building block of React application. It is a class or function which accepts an input and returns a React element.

	DOM node, and its desired properties.	
2.	It only holds information about the component type, its properties, and any child elements inside it.	It can contain state and props and has access to the React lifecycle methods.
3.	It is immutable.	It is mutable.
4.	We cannot apply any methods on elements.	We can apply methods on components.

5.	**Example:**
	const element =
	React.createElement(
	'div',
	{id: 'login-btn'},
	'Login'
	)
		Example:
		function Button ({ onLogin }) {
		return React.createElement(
		'div',
		{id: 'login-btn', onClick: onLogin},
		'Login'
		)
		}

35) How to write comments in React?

In React, we can write comments as we write comments in JavaScript. It can be in two ways:

1. Single Line Comments: We can write comments as /* Block Comments */ with curly braces:

```
{/* Single Line comment */}
```

2. Multiline Comments: If we want to comment more that one line, we can do this as

```
{ /*
  Multi
  line
  comment
*/ }
```

36) Why is it necessary to start component names with a capital letter?

In React, it is necessary to start component names with a capital letter. If we start the component name with lower case, it will throw an error as an unrecognized tag. It is because, in JSX, lower case tag names are considered as HTML tags.

37) What are fragments?

In was introduced in React 16.2 version. In React, Fragments are used for components to return multiple elements. It allows you to group a list of multiple children without adding an extra node to the DOM.

Example

```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  )
}
```

There is also a shorthand syntax exists for declaring Fragments, but it's not supported in many tools:

```
render() {
  return (
    <>
      <ChildA />
      <ChildB />
      <ChildC />
    </>
  )
}
```

For More Information, ***Click here***.

38) Why are fragments better than container divs?

- Fragments are faster and consume less memory because it did not create an extra DOM node.
- Some CSS styling like CSS Grid and Flexbox have a special parent-child relationship and add <div> tags in the middle, which makes it hard to keep the desired layout.
- The DOM Inspector is less cluttered.

39) How to apply validation on props in React?

Props validation is a tool which helps the developers to avoid future bugs and problems. It makes your code more readable. React components used special property PropTypes that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes.

We can apply validation on props using App.propTypes in React component. When some of the props are passed with an invalid type, you will get the warnings on JavaScript console. After specifying the validation patterns, you need to set the App.defaultProps.

```
class App extends React.Component {
          render() {}
}
Component.propTypes = { /*Definition */};
```

For More Information, ***Click here***.

40) What is create-react-app?

Create React App is a tool introduced by Facebook to build React applications. It provides you to create single-page React applications. The create-react-app are preconfigured, which saves you from time-consuming setup and configuration like Webpack or Babel. You need to run a single command to start the React project, which is given below.

```
$ npx create-react-app my-app
```

This command includes everything which we need to build a React app. Some of them are given below:

- It includes React, JSX, ES6, and Flow syntax support.
- It includes Autoprefixed CSS, so you don't need -webkit- or other prefixes.
- It includes a fast, interactive unit test runner with built-in support for coverage reporting.

- It includes a live development server that warns about common mistakes.
- It includes a build script to bundle JS, CSS, and images for production, with hashes and source maps.

For More Information, ***Click here***.

React Refs Interview Questions

41) What do you understand by refs in React?

Refs is the shorthand used for references in React. It is an attribute which helps to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

For More Information, ***Click here***.

42) How to create refs?

Refs can be created by using React.createRef() and attached to React elements via the ref attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.callRef = React.createRef();
  }
  render() {
    return <div ref={this.callRef} />;
  }
}
```

43) What are Forward Refs?

Ref forwarding is a feature which is used for passing a ref through a component to one of its child components. It can be performed by making use of the React.forwardRef() method. It is particularly useful with higher-order components and specially used in reusable component libraries.

Example

```
import React, { Component } from 'react';
import { render } from 'react-dom';

```

```
const TextInput = React.forwardRef((props, ref) => (
  <input type="text" placeholder="Hello World" ref={ref} />
));

const inputRef = React.createRef();

class CustomTextInput extends React.Component {
  handleSubmit = e => {
    e.preventDefault();
    console.log(inputRef.current.value);
  };
  render() {
    return (
      <div>
        <form onSubmit={e => this.handleSubmit(e)}>
          <TextInput ref={inputRef} />
          <button>Submit</button>
        </form>
      </div>
    );
  }
}
export default App;
```

For More Information, ***Click here***.

44) Which is the preferred option callback refs or findDOMNode()?

The preferred option is to use callback refs over findDOMNode() API. Because callback refs give better control when the refs are set and unset whereas findDOMNode() prevents certain improvements in React in the future.

```
class MyComponent extends Component {
  componentDidMount() {
    findDOMNode(this).scrollIntoView()
  }
  render() {
    return <div />
  }
}
```

The recommended approach is:

```
class MyComponent extends Component {
  componentDidMount() {
    this.node.scrollIntoView()
  }
  render() {
    return <div ref={node => this.node = node} />
  }
}
class MyComponent extends Component {
  componentDidMount() {
    this.node.scrollIntoView()
  }
  render() {
    return <div ref={node => this.node = node} />
  }
}
```

45) What is the use of Refs?

The Ref in React is used in the following cases:

- It is used to return a reference to the element.
- It is used when we need DOM measurements such as managing focus, text selection, or media playback.
- It is used in triggering imperative animations.
- It is used when integrating with third-party DOM libraries.
- It can also use as in callbacks.

For More Information, ***Click here***.

React Router Interview Questions

46) What is React Router?

React Router is a standard routing library system built on top of the React. It is used to create Routing in the React application using React Router Package. It helps you to define multiple routes in the app. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

For More Information, ***Click here***.

47) Why do we need a Router in React?

React Router plays an important role to display multiple views in a single page application. It is used to define multiple routes in the app. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular Route. So, we need to add a Router library to the React app, which allows creating multiple routes with each leading to us a unique view.

```
<switch>
    <h1>React Router Example</h1>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
</switch>
```

48) List down the advantages of React Router.

The important advantages of React Router are given below:

- In this, it is not necessary to set the browser history manually.
- Link uses to navigate the internal links in the application. It is similar to the anchor tag.
- It uses Switch feature for rendering.
- The Router needs only a Single Child element.
- In this, every component is specified in <Route>.
- The packages are split into three packages, which are Web, Native, and Core. It supports the compact size of the React application.

49) How is React Router different from Conventional Routing?

The difference between React Routing and Conventional Routing are:

SN	Conventional Routing	React Routing
1.	In Conventional Routing, each view contains a new file.	In React Routing, there is only a single HTML page involved.

2.	The HTTP request is sent to a server to receive the corresponding HTML page.	Only the History attribute <BrowserRouter> is changed.
3.	In this, the user navigates across different pages for each view.	In this, the user is thinking he is navigating across different pages, but its an illusion only.

50) Why you get "Router may have only one child element" warning?

It is because you have not to wrap your Route's in a <Switch> block or <div> block which renders a route exclusively.

Example

```
render((
  <Router>
    <Route {/* ... */} />
    <Route {/* ... */} />
  </Router>
)
```

should be

```
render(
  <Router>
    <Switch>
      <Route {/* ... */} />
      <Route {/* ... */} />
    </Switch>
  </Router>
)
```

51) Why switch keyword used in React Router v4?

The 'switch' keyword is used to display only a single Route to rendered amongst the several defined Routes. The <Switch> component is used to render components only when the path will be matched. Otherwise, it returns to the not found component.

React Styling Interview Questions

52) How to use styles in React?

We can use style attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in camelCased properties rather than a CSS string. The style attribute is consistent with accessing the properties on DOM nodes in JavaScript.

Example

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')'
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</div>
}
```

53) How many ways can we style the React Component?

We can style React Component in mainly four ways, which are given below:

- Inline Styling
- CSS Stylesheet
- CSS Module
- Styled Components

For More Information, ***Click here***.

54) Explain CSS Module styling in React.

CSS Module is a CSS file where all class names and animation names are scoped locally by default. It is available only for the component which imports it, and without your permission, it cannot be applied to any other Components. You can create CSS Module file with the .module.css extension.

For More Information, ***Click here***.

55) What are Styled Components?

Styled-Components is a library for React. It is the successor of CSS Modules. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS. It is scoped to a single component and cannot leak to any other element in the page.

The styled-components provides:

- Automatic critical CSS
- No class name bugs
- Easier deletion of CSS
- Simple dynamic styling
- Painless maintenance

For More Information, ***Click here***.

React Redux Interview Questions

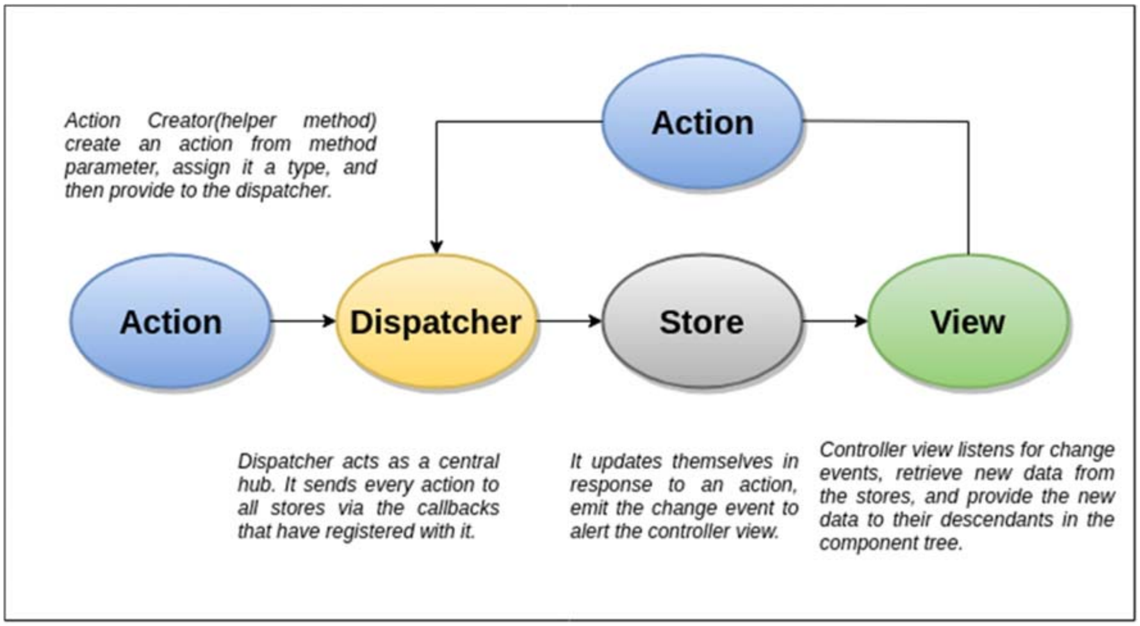
56) What were the major problems with MVC framework?

The major problems with the MVC framework are:

- DOM manipulation was very expensive.
- It makes the application slow and inefficient.
- There was a huge memory wastage.
- It makes the application debugging hard.

57) Explain the Flux concept.

Flux is an application architecture that Facebook uses internally for building the client-side web application with React. It is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model. It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.


Action Creator(helper method) create an action from method parameter, assign it a type, and then provide to the dispatcher.
Action
Action
Dispatcher
Store
View
Dispatcher acts as a central hub. It sends every action to all stores via the callbacks that have registered with it.
It updates themselves in response to an action, emit the change event to alert the controller view.
Controller view listens for change events, retrieve new data from the stores, and provide the new data to their descendants in the component tree.

For More Information, ***Click here***.

58) What is Redux?

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. The Redux application is easy to test and can run in different environments showing consistent behavior. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

For More Information, ***Click here***.

59) What are the three principles that Redux follows?

The three principles that redux follows are:

1. **Single source of truth:** The State of your entire application is stored in an object/state tree inside a single Store. The single State tree makes it easier to keep changes over time. It also makes it easier to debug or inspect the application.

2. **The State is read-only:** There is only one way to change the State is to emit an action, an object describing what happened. This principle ensures that neither the views nor the network callbacks can write directly to the State.
3. **Changes are made with pure functions:** To specify how actions transform the state tree, you need to write reducers (pure functions). Pure functions take the previous State and Action as a parameter and return a new State.

60) List down the components of Redux.

The components of Redux are given below.

- **STORE:** A Store is a place where the entire State of your application lists. It is like a brain responsible for all moving parts in Redux.
- **ACTION:** It is an object which describes what happened.
- **REDUCER:** It determines how the State will change.

For More Information, ***Click here***.

61) Explain the role of Reducer.

Reducers read the payloads from the actions and then updates the Store via the State accordingly. It is a pure function which returns a new state from the initial State. It returns the previous State as it is if no work needs to be done.

62) What is the significance of Store in Redux?

A Store is an object which holds the application's State and provides methods to access the State, dispatch Actions and register listeners via subscribe(listener). The entire State tree of an application is saved in a single Store which makes the Redux simple and predictable. We can pass middleware to the Store which handles the processing of data as well as keep a log of various actions that change the Store's State. All the Actions return a new state via reducers.

63) How is Redux different from Flux?

The Redux is different from Flux in the following manner.

SN	Redux	Flux

1.	Redux is an open-source JavaScript library used to manage application State.	Flux is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model.
2.	Store's State is immutable.	Store's State is mutable.
3.	In this, Store and change logic are separate.	In this, the Store contains State and change logic.
4.	It has only a single Store.	It can have multiple Store.
5.	Redux does not have Dispatcher concept.	It has single Dispatcher, and all actions pass through that Dispatcher.

64) What are the advantages of Redux?

The main advantages of React Redux are:

- React Redux is the official UI bindings for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.
- It encourages good 'React' architecture.
- It implements many performance optimizations internally, which allows to components re-render only when it actually needs.
- It makes the code maintenance easy.
- Redux's code written as functions which are small, pure, and isolated, which makes the code testable and independent.

65) How to access the Redux store outside a component?

You need to export the Store from the module where it created with createStore() method. Also, you need to assure that it will not pollute the global window space.

```
store = createStore(myReducer)
export default store
```