

COMPLETE NOTES ON

React.JS

COPYRIGHTED BY:-

INSTAGRAM :- @codes.learning

YOUTUBE :- Point of Programming

TELEGRAM :- codeslearning TG

WRITTEN BY :-

Mayuri Kharche (Software Engineer)

INDEX

| No | Topic | Page No |
|----|--------------------------------|---------|
| 1 | React Js Tutorial Introduction | 1 |
| 2 | React Features | 4 |
| 3 | Pros and Cons | 6 |
| 4. | React Js vs Angular Js | 9 |
| 5. | React Js vs React Native | 10 |
| 6. | React vs Vue | 12 |
| 7. | React JSX | 13 |
| 8- | React Components | 18 |
| 9. | React State | 22 |
| 10 | React Props | 25 |
| 11 | React Props Validation | 28 |
| 12 | React State vs Props | 31 |
| 13 | React Constructor | 32 |

| | | |
|----|----------------------------|----|
| 14 | React Component API | 35 |
| 15 | React Component Life Cycle | 40 |
| 16 | React Forms | 42 |
| 17 | Controlled Vs Uncontrolled | 46 |
| 18 | Conditional Rendering | 49 |
| 19 | React Lists | 51 |
| 20 | React Keys | 52 |
| 21 | React Refs | 53 |
| 22 | React Fragments | 56 |
| 23 | React Router | 58 |
| 24 | React CSS | 60 |
| 25 | React Animation | 66 |
| 26 | React Map | 68 |
| 27 | React Table | 70 |
| 28 | Higher Order Component | 71 |

| | | |
|----|----------------------|----|
| 29 | React Code Splitting | 73 |
| 30 | React Context | 75 |
| 31 | React Hooks | 77 |
| 32 | React Flux vs MVC | 78 |
| 33 | React Redux | 81 |
| 34 | React Portal | 82 |

React.Js

ReactJs is a declarative, efficient, and flexible Javascript library for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook.

Why we use React Js :-

The main objective of React Js is to develop User Interface (UI) that improves the speed of the apps. It uses virtual Dom (Javascript Object), which improves the performance of the app. The javascript virtual Dom is faster than the regular Dom. We can use ReactJs on the client and server-side as well as with other frameworks. It uses component and data patterns that improves readability and maintain larger apps.

• React create-react-app

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 code.

Requirement :-

The create React App is maintained by Facebook and can work on any platform for ex- macOS, Windows, Linux.

To create a React project using create react app you need to have installed the following things.

1] Node version ≥ 8.10

2] NPM version ≥ 5.6

- o Run the following code to check the Node version in command prompt.

```
$ 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>

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

```
$ npm -v
```

Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\javatpoint>npm -v
6.9.0
C:\Users\javatpoint>

In react application ,there are several files and folders in the root directory, some of them are as follows.

- 1] node-module :- 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, index.js and serviceWorker.js ,Here the App.js file always responsible for displaying the output screen react.
- 4] package-lock.json :- It is generated automatically for any operations where npm package modifiers either the node-module 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 projects.
- 6] Readme.md :- It provides the documentation to read about React topics

• React Features :-

• JSX :-

JSX stands for Javascript XML. It is a javascript syntax extension. It's an XML or HTML like syntax used by React JS. 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 React JS.

• Component :-

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

• One-way Data Binding :-

React JS is designed in such manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control over application. If the data flow is in another direction then it requires additional feature. 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 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 and new Dom. Once it has done, the real Dom will update only the things that have actually changed.

⑤ Simplicity :-

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

⑥ Performance:-

ReactJS is known to be a great performer. This feature makes it much better than other framework out there today. The reason behind this is that it manages virtual Dom. The Dom is a cross platform and programming API which deals with HTML, XML & XHTML. The Dom exists entirely in memory. Due to this when we create a component. We did not write directly to Dom.

Pros and Cons React Js:-

Advantage of React Js:-

① Easy to learn and use:-

→ React Js is much easier to learn and use. It comes with a good supply of documentation, tutorial and training resource.

② Creating Dynamic Web Application 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 JSX, which is a particular syntax letting HTML quotes.

③ Reusable Component:-

A React Js web application is made up of multiple components. And each component has its own logic and controls. These components are responsible for outputting small and reusable code helps to make your apps easier to develop and maintain.

④ Performance Enhancement:-

React Js 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. React Js solved this problem by introducing virtual Dom.

⑤ The Support of Handy Tools:-

React Js has also gained popularity due to the presence of handy set of tools. These tools make the task of developer understandable and easier. The React Developer Tool have been designed as chrome and Firebox dev extension and allow you to inspect the React component hierarchies in the virtual Dom.

⑥ Known to be SEO Friendly:-

Traditional Javascript framework have an issue in dealing with SEO. The search engines generally having trouble in reading JS application. Many web developers have often complained about this problem. React Js overcomes this problem that helps developers to be easily navigated on various search engines.

⑦ The Benefits of having Javascript Library:-

React Js is chosen by most of the web developers. It is because it is offering very rich Javascript library. The javascript library provides more flexibility to the web developers to choose the way they want.

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 continuously changes so fast, some of the developers not finding comfortable to relearn the new ways of doing things regularly.

② Poor documentation:-

It is another cons which are common for constantly updating technology. React technology is accelerating so fast that there is no documentation properly.

③ View Part:-

React Js cover only the UI layers of the app nothing else, so you still need to choose some other technologies to get complete tooling set for development in project.

④ JSX is barrier:-

React Js uses JSX. Its 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.

ReactJS vs AngularJS :-

| | 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 | Unidirectional |
| 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. |
| Performance | Slow | Fast, due to virtual DOM. |
| Best For | It is best for single page application that update a single view at a time. | It is best for single page applications that update multiple views at a time. |

React Js Vs React Native (Difference)

| | React Js | React Native |
|---|---|--|
| ① | The React Js initial release was in 2013. | The react Native initial release was in 2015. |
| ② | It is used for developing web applications. | It is used for developing mobile applications. |

web application

③ It can be executed on all platform

④ It uses a Javascript library and CSS for animations

⑤ It uses React-router for navigating web pages

⑥ It uses HTML tags

⑦ It can HTML tags

⑧ It can use code components , which saves a lot of valuable time

⑨ It provides high security

mobile applications

It is not platform independant. It takes more effort to be executed on all platform

It comes with built-in animation libraries.

It has built-in animation libraries.

It does not use HTML tags.

It does not use HTML tags

It can reuse React Native UI component & modules which allow hybrid apps to render natively.

It provides low security in comparison to ReactJS

React Vs Vue

| | React | Vue. |
|--------------------|--|--|
| Defination | React is declarative efficient , flexible open-source JS library for building reusable UI Component | Vue is an Open-source JS library for building reusable user interfaces and single page application |
| 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 & Insta. | Vue was created by Evan You, a former employee of Google working on many Angular project. He wanted to better version of Angular, just extracting the part which he liked about Angular and making it lighter. |
| Preferred language | Javascript / XML | HTML / Javascript |
| Size | The size of the React library is 100 kilobytes | The size of the Vue library is 60 kb. |

| | | |
|-------------------|---|--|
| Performance | The performance is slow as compared to Vue | Its performance is fast as compared to React. |
| Flexibility | React provides great flexibility to support third party library | Vue provides limited flexibility as compared to 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 support. | It is not suitable for long term support. |

React JSX

JSX provides you to write HTML/XML like structure in the same file where you write javascript code, then preprocessor will transform these expression into actual Javascript code. Just like XML/HTML, JSX tags name, attributes, and children.

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 component that contain both. We will learn component

- It is type-safe, and most of the errors can be found at compilation time.
- It makes easier to create template.

Nested Elements in JSX:-

APP. JSX

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

JSX Attribute :-

JSX use attributes with the HTML elements same as regular HTML. JSX uses convention for attribute rather than standard naming convention of HTML

Such as a class in HTML becomes className in JSX because the class is the reversed keyword in javascript. We can also user our own custom attribute, we need to use data-prefix.

Example:-

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

In JSX, We can specify attribute values in 2 ways;

① As String Literal:-

We can specify the values of attributes in double quotes.

```
var element = <h2 className = "firstAttribute" >
  Hellow 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;
```

2] Its Expressions:-

We can specify the values of attributes as expression using curly braces {}.

Example.

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

Export default App;

JSX Comments

JSX allows us to comments that begins with /* and ends with */ and wrapping them in curly braces {} just like in the case of JSX expressions

Example

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

JSX Styling

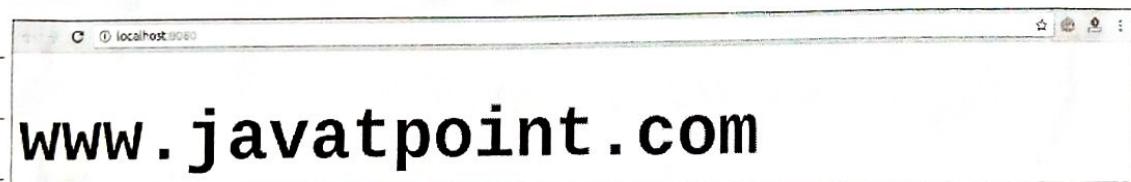
React always recommended to use inline style. To set inline styles, you need to use camelCase syntax. React automatically allows appending px after the number value on specific elements.

Example :-

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    var myStyle = {
      fontSize: 80
    }
  }
}
```

```
font-family: 'Courier',  
    color: '#003300'  
}  
return (  
    <div>  
        <h1 style={mystyle}> www.javatpoint.com </h1>  
    </div>  
);  
}  
}  
export default App;
```

Output:-



React Components

React Component



Functional

Component

Class

Component

Functional Component:-

In React, function component are a way to write component that only cont

a render method and don't have their own state. They are simply JavaScript function that may or may not receive data as a parameter. We can create function that takes props as input and returns what should we rendered.

```
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.

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>  
        );  
    }  
}
```

3:

}

```
class Second extends React.Component {  
    render() {  
        return (

## www.JavaPoint



This website contains the great CS tutorial.

)  
    }  
    export default App;
```



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 component.

Example:-

```
import React { Component } from 'react'  
class App extends React.Component {  
    constructor() {
```

```
Super(); and it helps in reusing state if  
this.state = {initial value of state if  
data: [array of objects] state  
[  
  {  
    "name": "Abhishek"  
  }  
  {  
    "name": "Saharsh"  
  }  
  {  
    "name": "Ajay"  
  }  
]  
} 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;
```

React State :-

The state is an updated structure that is used to contain data or information about the component. The state in component can change over time. The state in change over time happen as a response to user action or system event.

A state must be kept as simple as possible. It can be state by using the `setState()` method and calling `setState()` method triggers UI Updates. A state represents the component's local state or information. To state an initial state before any interaction occurs we need to use `getInitialState` method.

Defining State:-

To define a state, you have to first declare a default set of values for the defining the components initial state.

Example:-

```
import React, {Component} from 'react';
class App extends React.Component {
  constructor() {
    super();
    this.state = {displayBio: true};
  }
}
```

```
render() {
```

```
  const bio = this.state.displayBio ?
```

```
    <div>
```

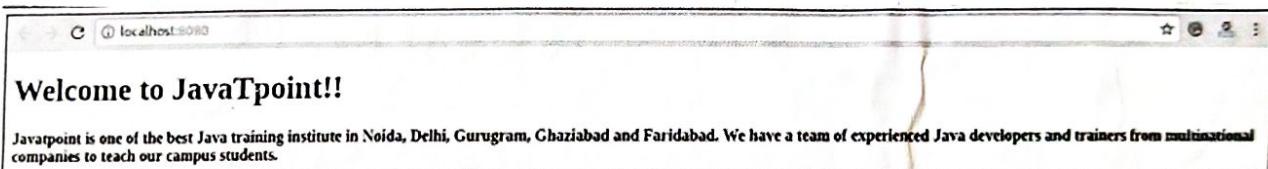
```
<h3> Javatpoint is one of the best training
```

institute in Noida, Delhi, Gurugram, Ghaziabad

```

</h3> </p>
</div>
): null;
return (
<div>
  <h1> Welcome to javaTpoint </h1>
  {bio}
</div>
);
}
export default App;

```



Changing the state:-

We can change the component state by using `setState()` method and passing a new state object as the argument. Create a new method `toggleDisplayBio()` in the above example.

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

```

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);
```

{

```
this.toggleDisplayBio() {
  this.setState({displayBio: !this.state
displayBio});
```

}

```
render() {
```

```
return (
```

```
<div>
```

```
<h1> Welcome to Javatpoint </h1>
```

{

```
this.state.displayBio ? (
```

```
<div>
```

```
<h4> Javatpoint is one of the best Java
institute in Noida </h4> </p>
```

```
<button onClick={this.toggleDisplayBio}> Show Less </button>
```

```
</div>
```

```
+<div> Click here to like our Facebook page </div>
```

```
<div>
```

```
<button onClick={this.toggleDisplayBio}>
  Read More </button>
```

```
<div>
```

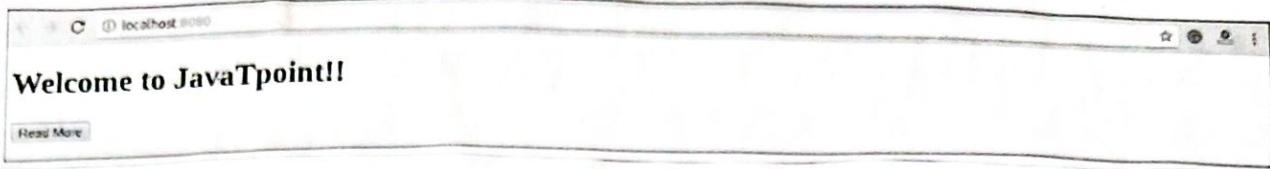
```
) {
```

```
</div>
```

```
) } } }
```

```
export default App;
```

when you click on Read More



React Props:-

Props stand for Properties. They are read-only components. It is an object which stores the value of attribute of a tag and works similar to the HTML attribute.

Props are immutable so we cannot modify this props from inside the component.

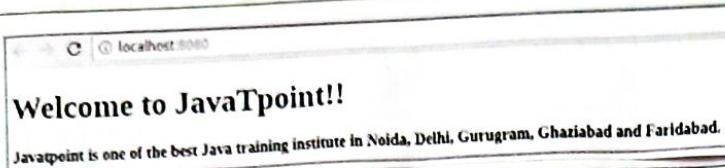
Example:-

```
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
          training institute in India </h4> </p>
      );
    }
}
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');



Default Props:-

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

Example

App.js

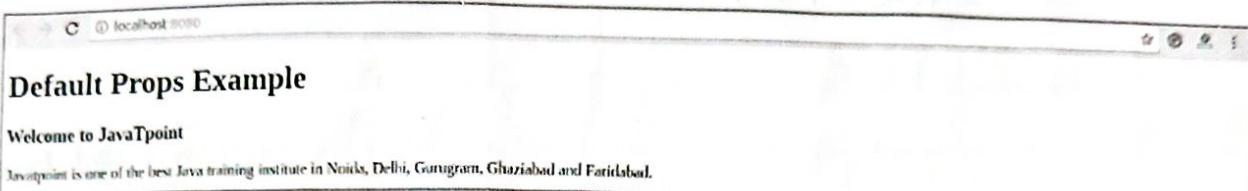
```
import React, {Component} from 'react';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1> default Props Example </h1>
        <h3> Welcome to {this.props.name} </h3>
        <p> JavaTpoint is one of the best Java
        training </p>
      </div>
    );
  }
}

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'));
```



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.

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 jtpProps={this.state.name}>
      </JTP>
    )
  }
}
```

```

</div>
}; }
class JTp extends React.Component {
  render() {
    return (
      <div>
        <h1> State & Prop example </h1>
        <h3> Welcome to {this.props.jtpProp} </h3>
        <p> JavaTpoint is one of the best Java
          training institute in Noida </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'));

```

State & Props Example

Welcome to JavaTpoint

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

React Props Validation:-

Props are an important mechanism for passing the read-only attribute to React component. The props are usually required to use correctly in the component.

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.

Syntax:-

```
class App extends React.Component {
  render() {}  
}
```

`Component.propTypes = { /* prop types */ }`

ReactJS Props Validator

| Props Type | Description. |
|---------------------------------|------------------------------------|
| ① <code>PropTypes.any</code> | The props can be of any data type. |
| ② <code>PropTypes.array</code> | The props should be an array. |
| ③ <code>PropTypes.bool</code> | The props should be a boolean. |
| ④ <code>PropTypes.func</code> | The props should be a function. |
| ⑤ <code>PropTypes.number</code> | The props should be a number. |
| ⑥ <code>PropTypes.Object</code> | The props should be an object. |

| | | |
|---|---|---|
| ⑦ | PropTypes.string | The props should be String |
| ⑧ | PropTypes.Symbol | The props should be Symbol |
| ⑨ | PropTypes.instanceOf | The props should be an instance of particular Javascript class. |
| ⑩ | PropTypes.isRequired | The props must be provided |
| ⑪ | PropTypes.element | The props must be element |
| ⑫ | PropTypes.node | The props can render anything : number, string, element or an array 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 type |

• React Js Custom Validators.

React Js allows creating a custom validation function to perform custom validation. The following arguments is used to create a custom validation function.

props:- It Should be first argument in the Component.

propName:- It is the propName that going to validate.

Component Name:- It is the componentName that are going to validate again.

Example:-

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

Difference between State and Props.

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

| | |
|------------------------|---------------------|
| by the child component | by child components |
|------------------------|---------------------|

Props are used to communicate between component

State can be used for rendering dynamic changes with the component

Stateless component can have Props

Stateless component cannot have State

Props make component reusable

State cannot make components reusable

Props are external and controlled by whatever render the component

The State is internal and controlled by the React Component itself.

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.

What is Constructor:-

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

class.

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

In react, constructor are mainly used.

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

Example :-

App.js

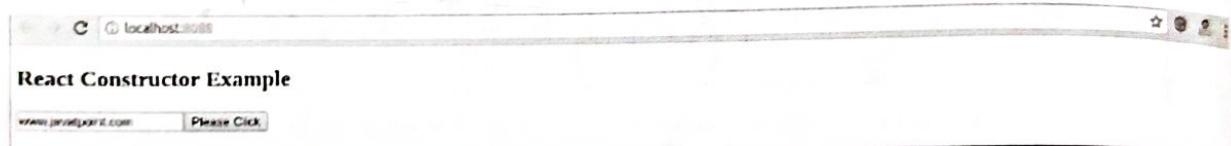
```
import React, {Component} from 'react';  
class App extends Component {  
    constructor(props)  
        Super(props);  
        this.state = {  
            data : "www.javatpoint.com"  
        }  
        this.handleEvent = this.handleEvent.bind(this);  
    } handleEvent () {  
        console.log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

```
</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'));
```



Arrow Function:-

The arrow function is the new feature in ES6 standard. If you need to use arrow functions. It is not necessary to bind any event to 'this'.

```
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 </h2>  
            <input type="text" value={this.state.data} />  
            <button onClick={this.handleEvent}>  
                Please Click </button>  
        </div>  
    );  
}  
export default App;
```

React Component API.

React JS component is a top-level API. It makes the code completely individual and reusable in the application. It includes various methods for creating element.

- o Transforming element.
- o Fragments

① setState()

This method is used to update state of the component. This method does not always replace the state immediately. It only adds change to the original state. It is primary method that is used to update the user interface (UI) in response (UI) in event handler and server response.

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

```

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: "It's best tutorial"
    });
  }
  render() {
    return (
      <div>
        <h1> {this.state.msg} </h1>
        <button onClick={this.updateState}>
          SetSTATE </button>
      </div>
    );
  }
}
export default App;

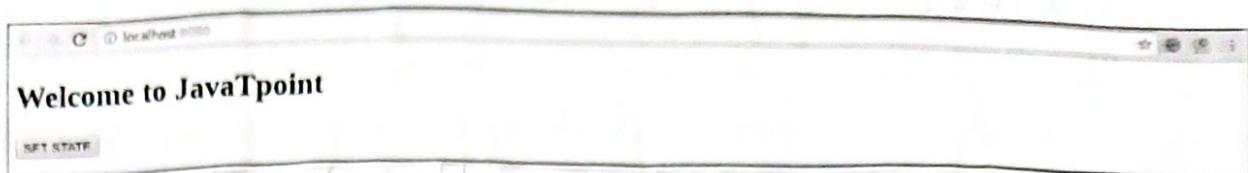
```

Main.js

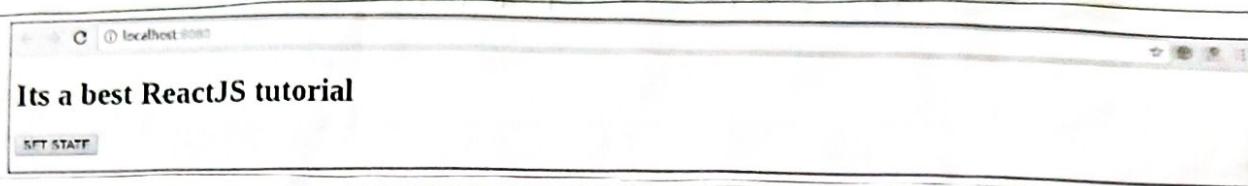
```

import React from 'react';
import ReactDOM 'react-dom';
import App from '/App.js';
ReactDOM.render(<App/>, document.getElementById('app'));

```



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



forceUpdate()

This method allows us to update the component manually.

Component.forceUpdate(callback);

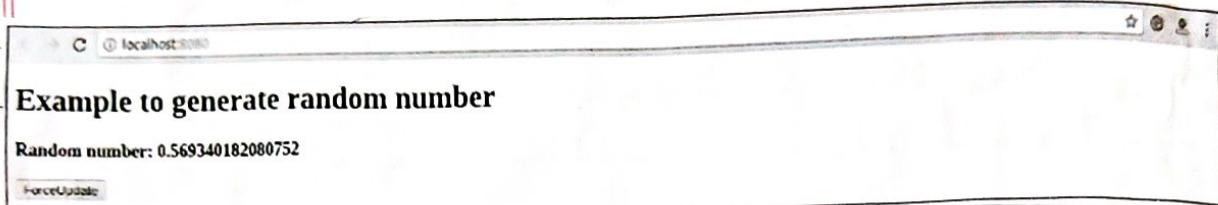
Example - App.js.

```
import React, {Component} from 'react';
class App extends React.Component {
  constructor() {
    Super();
    this.forceUpdateState = this.forceUpdateState.bind(this);
  }
  forceUpdateState() {
    this.forceUpdate();
  }
  render() {
    return (
      <h1>It's a best ReactJS tutorial</h1>
    );
  }
}
```

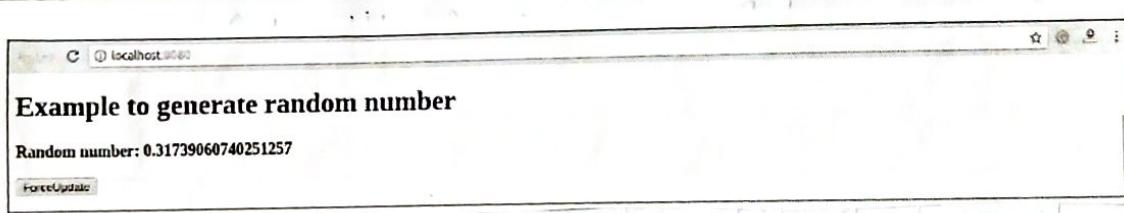
```

<div>
  <h1> Example to generate random Number </h1>
  <h3> Random number : {Math.random()} </h3>
  <button onClick = {this.forceUpdateState}>
    ForceUpdate </button>
  </div>
); >
export default App;

```



Each time when you click on ForceUpdate button.



`findDOMNode()`

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

`ReactDOM.findDOMNode(component);`

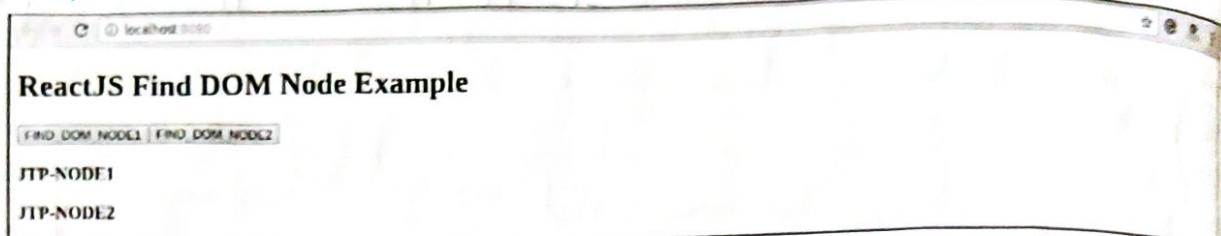
App.js

```
import React {Component} from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component{
  constructor () {
    Super ();
    this.findDOMNodeHandler1 = this.findDOMNode-
      Handler1.bind(this);
    this.findDOMNodeHandler2 = this.findDOMNode-
      Handler2.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-NODE 1 </button>
        <button onClick = {this.findDOMNodeHandler2}>
          FIND-DOM-NODE 2 </button>
        <h3 id = "myDivOne"> JTP-Node1 </h3>
    )
  }
}
```

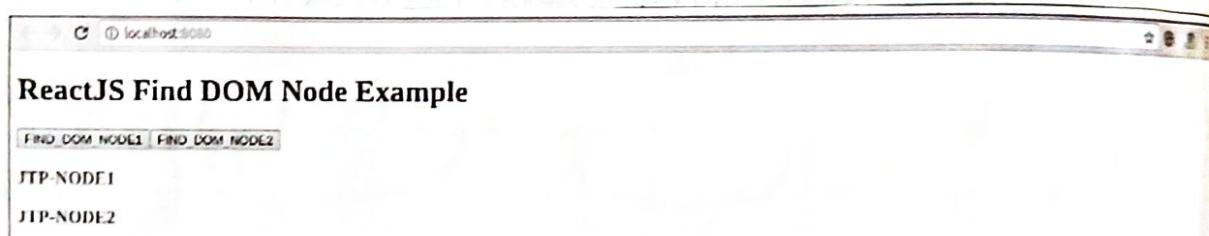
```

<button onClick = {this.forceUpdateState}>
  ForceUpdate </button>
</div>
);
}
export default App;

```



Once you click on the button, the color of the node gets changed.



React Component Life-Cycle

In React Js , every component creation process involves various lifecycle methods. These lifecycle methods are termed as component lifecycle. These lifecycle methods are not very complicated and called at various during a component's life.

Initial Phase → Mounting phase → Updating phase → Unmounting phase.

① Initial Phase -

- get defaultProps()

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.

- getInitialPhase()

It is used to specify the default value of this.state. It is invoked before the creation of the component.

② Mounting Phase

- ComponentWillMount()

This is invoked immediately before a component gets rendered into the Dom. In the case when you call setState() inside this method

- ComponentDidMount()

This is invoked immediately after component gets rendered and placed on the Dom. You can do any Dom querying operations.

- render()

This method is defined in each and every component. It is responsible returning a single root HTML node element.

③ Updating Phase

- ComponentWillReceiveProps()

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()

• ShouldComponentUpdate()

It is invoked when a component decides updating occurs.

• ComponentWillUpdate()

It is invoked just before the component updating occurs. Here you can't change the component state by invoking this.setState() method.

• ComponentDidUpdate()

It is invoked immediately after the component updating occurs. In this method, you can put any code inside this.

4] Unmounting Phase

ComponentWillUnmount()

This method is invoked immediately before a component is destroyed permanently. It performs necessary cleanup.

React Forms:-

Forms are an integral part of any modern web application. It allows the users to interact with the applications as well as gather information from the user. Forms can perform many tasks that depends on the nature of your business requirements and

logic such as authentication of the user.

Creating Form :-

React offers a stateful, reactive approach to build a form. The component rather than Dom usually handles the React form.

① Uncontrolled Component.

② Controlled component.

① Uncontrolled Component :-

The uncontrolled component input is similar to the traditional HTML form input. The Dom itself handles the form data. The HTML element maintain their own state that will be updated when the input values changes. To write an uncontrolled component.

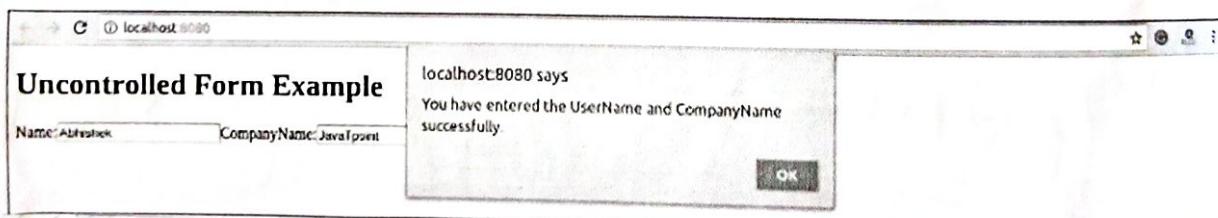
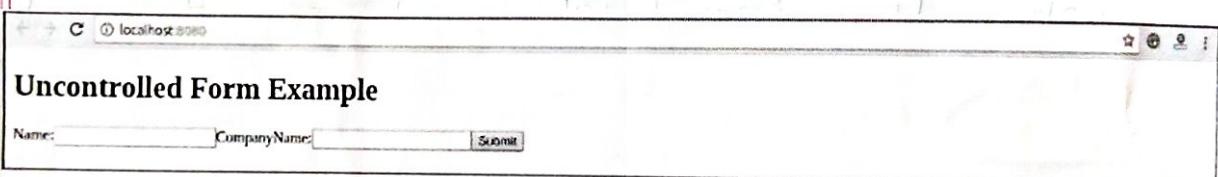
```
import React, {Component} from 'react';
class App extends React.Component{
  constructor(props)
    super(props);
    this.updateSubmit = this.updateSubmit.bind(this);
    this.input = React.createRef();
}
updateSubmit(event){
  alert("You have entered the username and CompanyName successfully");
  event.preventDefault();
}
render(){}
```

```

    return (
      <form onSubmit={this.updateSubmit}>
        <h1> Uncontrolled Form </h1>
        <label> Name:</label>
        <input type="text" ref={this.input}>
        </label>
        <label>
          Company Name:</label>
        <input type="text" ref={this.input}>
        </label>
      </form>
    );
  }
}

export default App;

```



Controlled Component

Controlled component have functions that govern the data passing into them on every onChange event; rather than grabbing the data only ones.

```
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 </h1>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange}>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
export default App;
```

Controlled Component Vs Uncontrolled Component.

Controlled

It does not maintain internal state

Data is controlled by parent component

It accept the current value as a prop

It allows validation control

It has better control over the form elements and data

Uncontrolled

It maintains internal state.

Data is controlled by a Dom itself

It uses a ref for their current values.

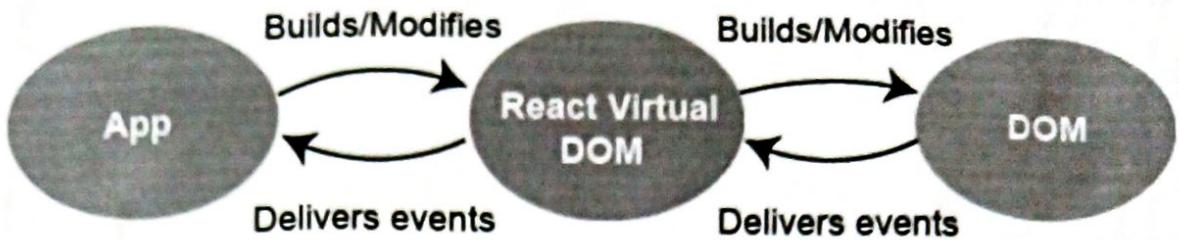
It does not allow validation control.

It has limited control over the form and elements.

React Events -

React has its own event handling system which is very simillar to handling events on Dom elements. The react event handling system is known as Synthetic Events. The Synthetic events is a across browser wrapper of the browser native event.

- ① React events are named as camelCase instead of lowercase.



- ② A function is passed as a event handler instead of a string.

For declaration in plain HTML:-

```

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

Event declaration in React:-

```

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

- ③ In react, we cannot return false to prevent the default behaviour we must call preventDefault event explicitly to prevent the

Example:-

```

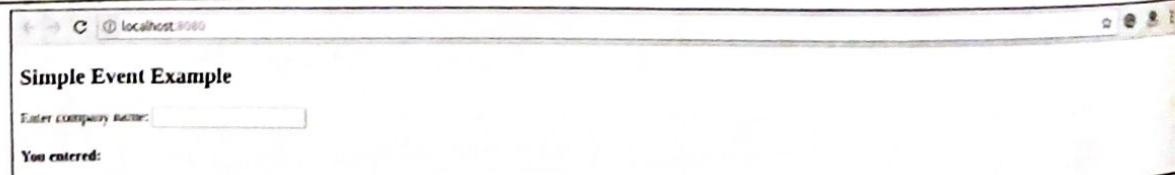
import React,{Component} from 'react'
class App extends React.Component{
  constructor(props)
    Super(props)
    this.state = {
      CompanyName =
    };
  
```

```
}; }

changeText (event) {
  this.setState({
    company Name: event.target.value
  });
}

render () {
  return (
    <div>
      <h2> Simple Event </h2>
      <label html For='name'> Enter Company
        Name </label>
      <input type="text" id="company Name"
        onchange = {this.changeText bind (this)}
        <h4> You entered : {this.state.company
          Name} </h4>
    </div>
  );
}
}

export default App;
```



React Conditional Rendering:-

There is more than one way to do conditional rendering in React.

- if

- It is the easiest way to have conditional rendering in React in the `render` method.

```
function UserLogin (props) {  
    return <h1> Welcome Back </h1>;  
}
```

```
function GuestLogin (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.

The operator is used to checking the condition.

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

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

```
function Example()
```

```

return(<div>
  { (10) && alert('This alert will be
    Shown!')
  }
</div>
);
}

```

Ternary Operator :-

```

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

```

Switch Case Operator

```

functional 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.

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

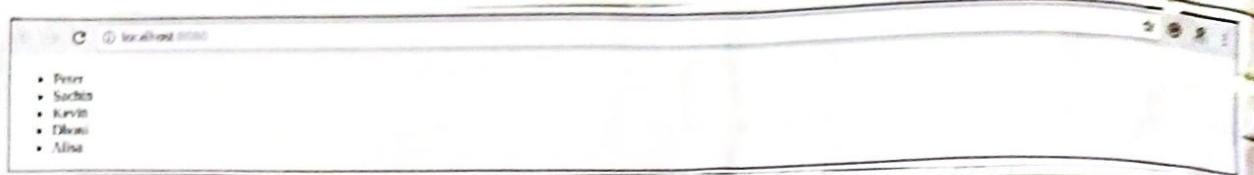
React Lists:

Lists are used to display in an ordered form and mainly used to display menus on websites. The map() function is used to traversing the list. Include the new list `` elements and render it to dom.

```
import React from 'react';
import ReactDOM from 'react-dom';
const myList = ['Peter', 'Sachin', 'Kelvin', 'Dhoni', 'Alisa'];
const myListItems = myList.map((myList) => {
  return <li>{myList}</li>
});
ReactDOM.render(
  <ul>{listItems}</ul>
  , document.getElementById('root')
);
```

```
document.getElementById('app')  
);  
export default App;
```

Output:



React Keys:-

A key is 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 component or when the user alert the lists.

It also helps to determine which component in a collection need to be rendered instead of re-rendering the entire set of components every time.

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

```
const updatedlists = stringlist.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 of the lists. It can be shown in the below,

```
const stringLists = ['Peter', 'Suchin', 'Kevin', 'Dhoni',  
    'Alisa'];  
const updatedLists = stringLists.map((strList, index) =>  
{  
    <li key={index}> {strList} </li>  
});
```

React Refs:-

Refs is the shorthand used for references 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.

When to Use Refs:-

- When we need DOM measurements such as managing focus, text selection, or media playback.
- It is used 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 identifying anything that can be done declaratively 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 Element via the `ref` attribute. It is commonly assigned to an instance property when a component is created and them 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 Ref's

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;
```

Ref current Properties

- 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 component because they don't have instances.

Callback Refs :-

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 passing a callback function to the ref attribute of a component.

```
<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.

```
this.callRefInput.value
```

React with useRef()

It is introduced in React 16.7 and above version. It helps to get access the DOM node element and we can interact with the DOM node or element such as focusing the element or accessing the input element value. It returns the ref object whose .current property initialized to the passed argument.

```
const refContainer = useRef(initialValue);

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

React Fragments:-

The render method can return single elements or multiple elements. The render method will only render a single node inside at a time.

```
class App extends React.Component {
    render() {
        return (
            <div>
                <h2> Hello World </h2>
                <p> Welcome to the JavaTpoint </p>
            </div>
        );
    }
}
```

Syntax :-

```
<React.Fragment>
  <h2> Child 1 </h2>
  <p> Child 2 </p>
  ...
</React.Fragment>
```

Why we use Fragments?

- (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 '<>' instead of the 'React.Fragment'

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

Key Fragments:-

The shorthand syntax does not accept key

key attribute. You need a key for mapping a collection to an array of fragments such as to create a description list.

```
Function = (props) {  
    return (  
        <Fragment>  
        { props.item.data.map(item => (  
            <React.Fragment key={item.id}>  
                <h2>{item.name}</h2>  
                <p>{item.url}</p>  
                <p>{item.description}</p>  
            </React.Fragment>  
        ))}  
        </Fragment>  
    )  
}
```

React Router:-

Routing is a process in which a user is directed to different pages based on their action or request. React.js Router mainly used for developing Single Page application. React Router is used to define multiple routes in the application.

React Route is a standard library system built on the 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.

React contain 3 different package of routing

① react-router :- It provides the core routing components and function for the React Router application.

② react-router-native :-

It is used for mobile application.

③ react-router-dom :-

It is used for web application design.

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

Components in React Router.

<BrowserRouter>

- It is used for handling URL

<HashRouter>

- It is used to handle static request.

Benefits of React Router:-

① It is not necessary to set the browser history manually.

② Link uses a navigate the internal links in the application. It is similar to anchor tag.

③ It uses switch feature for rendering

④ If the router needs only a single child element.

React CSS

CSS in React is used to style the React App or component. The style attribute is the most common attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a javascript object in camelcased properties rather than CSS string.

① Inline Styling

The inline styles are specified with a javascript object in camelcase version of the style name. Its value is the styles value which we usually take in a string.

App.js

```
import React from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1 style={{color:"Green"}}>Hello JavaT&lt;/h1>
        <p>Here you can find all CS Tutorials</p>
      </div>
    );
  }
}
export default App;
```

② CSS Stylesheet :-

You can write styling in a separate file for your React application, and save the file with a CSS extension. You can import the file in your application.

App.js.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';
class App extends React.Component {
  render() {
    <div>
      <h1> Hello JavaTpoint </h1>
      <p> You can find all cs tutorials </p>
    </div>
  }
}
export default App;
```

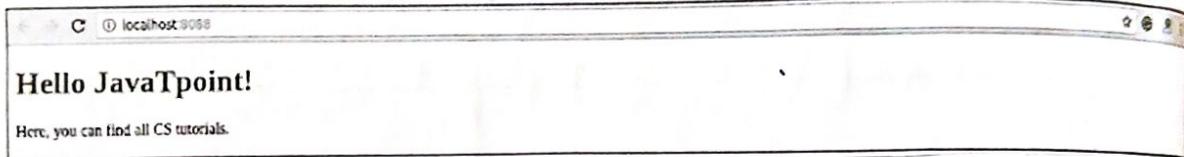
App.css

```
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>
```



③ CSS Module

CSS Module is another way of adding styles to your application. In 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.

App.js

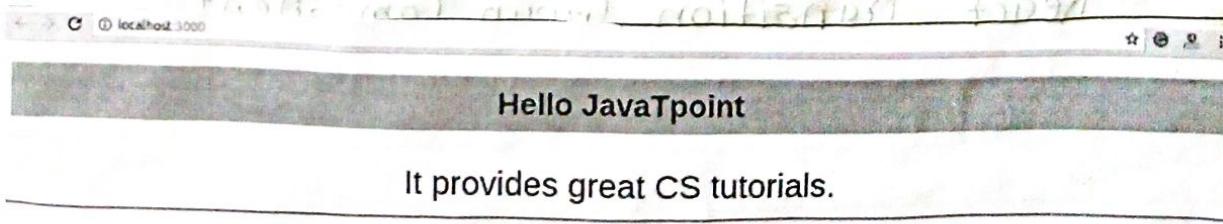
```
import React from 'react';
import ReactDOM from 'react-dom';
import styles from './myStyles.module.css'
```

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

myStyles.module.css

```
• myStyle {  
  background-color: #cdcdc0;  
  color: red;  
  padding: 10px;  
  font-family: Arial;  
  text-align: center;  
}  
• parastyle {  
  color: green;  
  font-family: Arial;  
  font-size: 35px;  
  text-align: center;  
}
```

}



It provides great CS tutorials.

4. Styled Component:-

Styled component is a library for React. It uses enhanced CSS for styling React components in your application which is written with a mixture of JavaScript and CSS.

The styled component provides.

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

React Animation.

The animation is a technique in which images are manipulated to appear as moving images. It is one of the most used techniques to make interactive web applications.

React Transition group has mainly 2 API Create Transition.

① React Transition Group:-

It uses as low-level API for animation.

② React CSS Transition Group:-

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

React Transition Group Component.

Provides 3 main components.

① Transition

② CSS Transition

Transition Group.

Transition:- It has a simple component API to declare 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 state as well.

- ① entering
- ② entered.
- ③ existing
- ④ exited.

CSS Transition:-

The CSS transition component uses 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.

- o Appear
- o Enter
- o Exist.

Transition Group:-

This component is used to manage a set of transition component in a list. It is a state machine that controls the mounting and unmounting components overtime. The transition component does not define any animation directly. 'Transition Group' component can have different animation with a component.

App.js

```
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 Examples </h1>
        {items}
      </div>
    );
  }
}
```

```
<button onClick={this.handleAdd}> Insert them
    </button>
<CSSTransitionGroup
    transitionName="example"
    transitionEnterTimeout={800}
    transitionLeaveTimeout={600}>
    {items}
    </CSSTransitionGroup>
</div>
);
}
}

export default App;
```

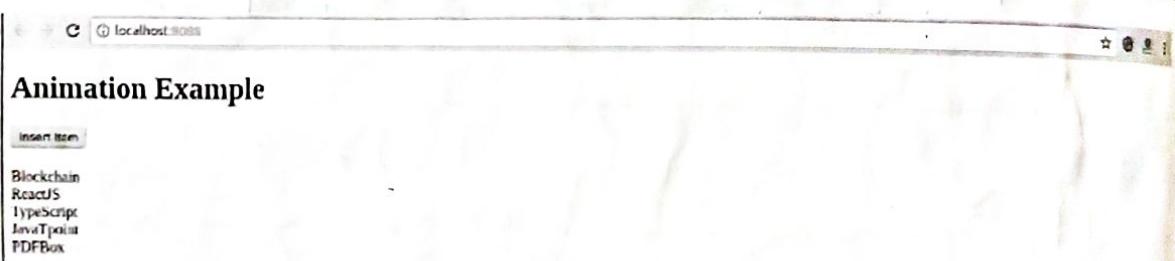
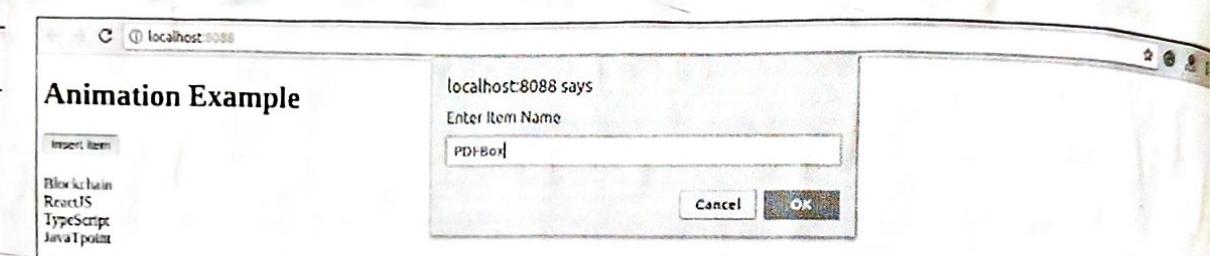
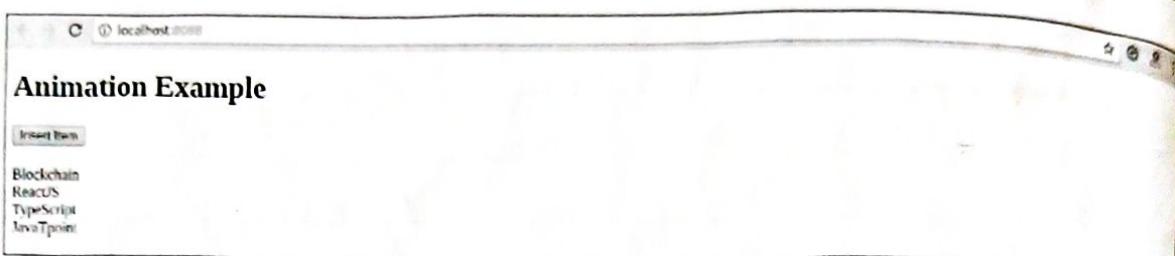
Main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';
ReactDOM.render(<App />, document.getElementById
('app'));
```

Style.css

```
.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;  
}
```



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 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 for:

① Traversing the list element.

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

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

```
function NameList(props) {
```

```
  const myList = (props) {
```

```
    const listItems = myList.map((myList) =>
```

```
      <li> {myList} </li>
```

```
    );
```

```
    return (
```

```
      <div>
```

```
        <h2> React Map example </h2>
```

```
        <ul> {listItems} </ul>
```

```
      </div>
```

```
    );
```

```
  const myLists = ['A', 'B', 'C', 'D'];
```

```
  ReactDOM.render(
```

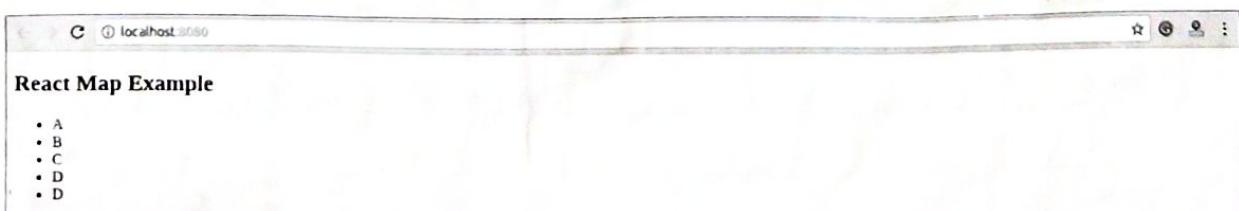
```
    <NameList myList = {myLists} />
```

```
    document.getElementById('app')
```

```
  );
```

```
export default App;
```

Output -



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 lightweight, fast, fully customizable and extendable Datagrid built for React. It is fully controllable via optional props and callbacks.

Features:-

- ① It is lightweight at 11 kb.
- ② It is fully customizable.
- 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

```
import React,{Component} from 'react';
import ReactTable from "react-table";
import "react-table/react-table.css";
class App extends Component {
  render() {
    const data = [
      {name : "Ityaan", age : 26},
      {name : "Ahana", age : 22}
    ];
  }
}
```

```
name: 'Peter',
```

```
age: 40
```

```
}, {
```

```
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;
```

React Higher-Order Component

It 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.

```
const NewComponent = higherOrderComponent  
(wrappedComponent)
```

HOC.JS

```
import React, {Component} from 'react';
export default function HOC(HOCComponent){
    return class extends Component {
        render(){
            return (
                <div>
                    <HOCComponent>
                </div>
            );
        }
    }
}
```

App.JS

```
import React, {Component} from 'react';
import HOC from './HOC';
class App extends Component {
    render(){
        <div>
            <h2> HOC Example </h2>
            JavaTPoint provides best CS Tutorial
        </div>
    }
}
App = HOC(App);
export default App;
```

Output -



HOC Example

JavaTPoint provides best CS tutorials.

React Code Splitting :-

The react app bundled their files using tools like Webpack or Browserify. Bundling is a process which takes multiple files and merge them into a single file, which is called bundle. The bundle is responsible for loading an entire app at ones on the webpage.

Code splitting improves:-

- (1) The performance of the app.
- (2) The impact on memory.
- (3) The downloadable kilobytes 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>  
            <Example Component/>  
        </div>  
    );  
}
```

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.

```
import MyErrorBoundary from '/MyErrorBoundary'  
const ExampleComponent = React.lazy(() =>  
    import ('/Example Component')  
);  
const ExampleComponent = React.lazy(() =>  
    import ('/Example Component')  
);  
const MyComponent = () => (  
    <div>  
        <MyErrorBoundary>  
            Suspense fallback = {<div>Loading</div>}  
            <section>  
                <ExampleComponent/>  
                <ExampleComponent/>  
            </section>  
        </suspense>  
    </MyErrorBoundary>  
    </div>  
);
```

React Context:-

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

React Context API :-

- ① React.createContext
- ② Context.Provider
- ③ Context.Consumer
- ④ class.contextType

① React.createContext :-

It creates a context object

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

② Context.Provider :-

Every Context object has a provider React component which allows consuming components to subscribe to context changes.

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

③ Context.Consumer :-

It is an React Component which subscribes to the context changes. It allows us to the context within an function.

```
<MyContext.Consumer>
```

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

Class context Type:-

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 using this `context`.

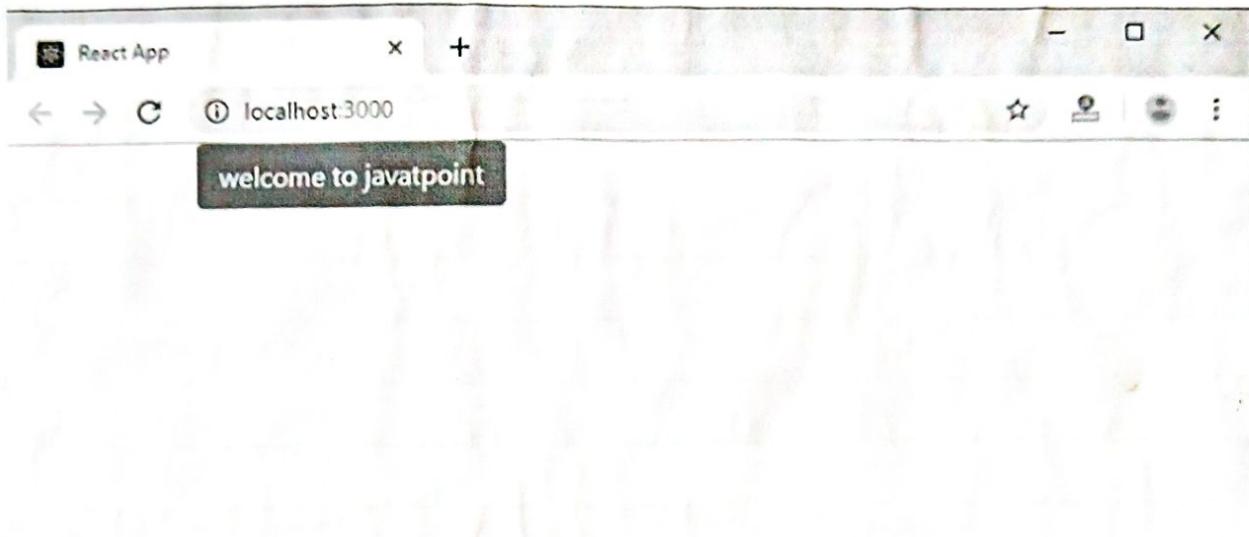
```
import React, {Component} from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
const BtnColorContext = React.createContext('
  btn btn-dark yellow');

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

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

class ThemedButton extends Component {
  static contextType = BtnColorContext;
  render() {
    return <button className={this.context}>
      Welcome to JavaTpoint
    </button>
  }
}
```

```
</button>
}
}
export default App;
```



React Hooks:-

Hooks are the new feature introduced in React 16.8 version. It allows you to use state and other React feature without writing a class.

Rules of Hooks:-

- ① Only call Hooks at the top level
- ② Only call Hooks from React functions.

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.

App.js

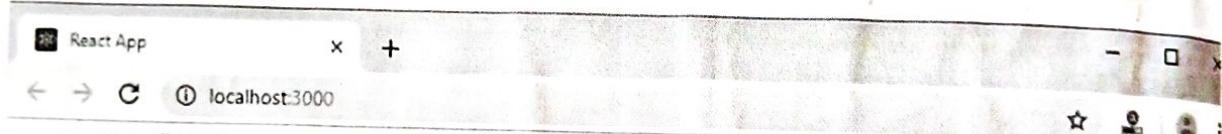
```
import React, {useState} from 'react';
```

```

function CountApp() {
  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

```



React Flux Vs MVC

MVC

MVC stands for Model View Controller. It is an architectural pattern used for developing the user interface.

MVC Architecture:-

Model:- It is responsible for maintaining the behaviour and data of an application.

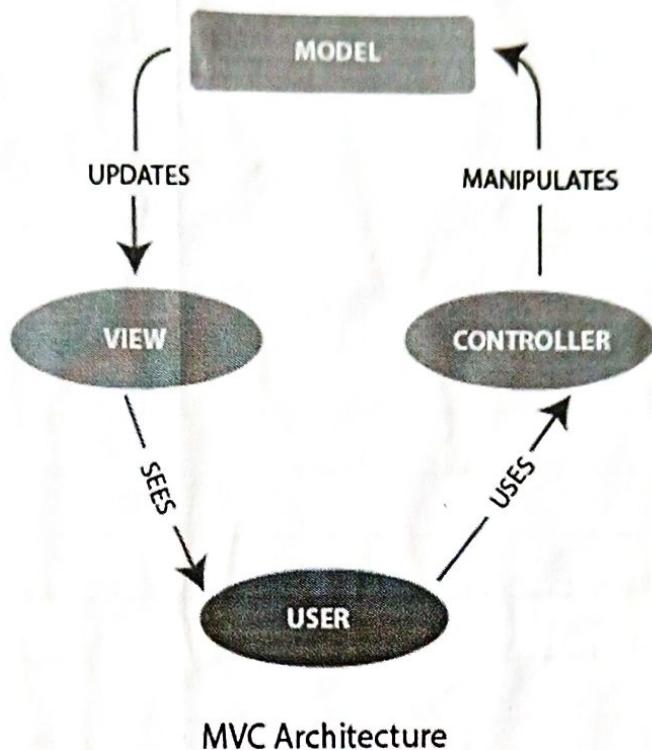
View:- It is used to display the model in the user interface.

Controller:- It acts as an interface between

the Model and the view Components. It takes user input, manipulates the data and causes View

You clicked 4 times.

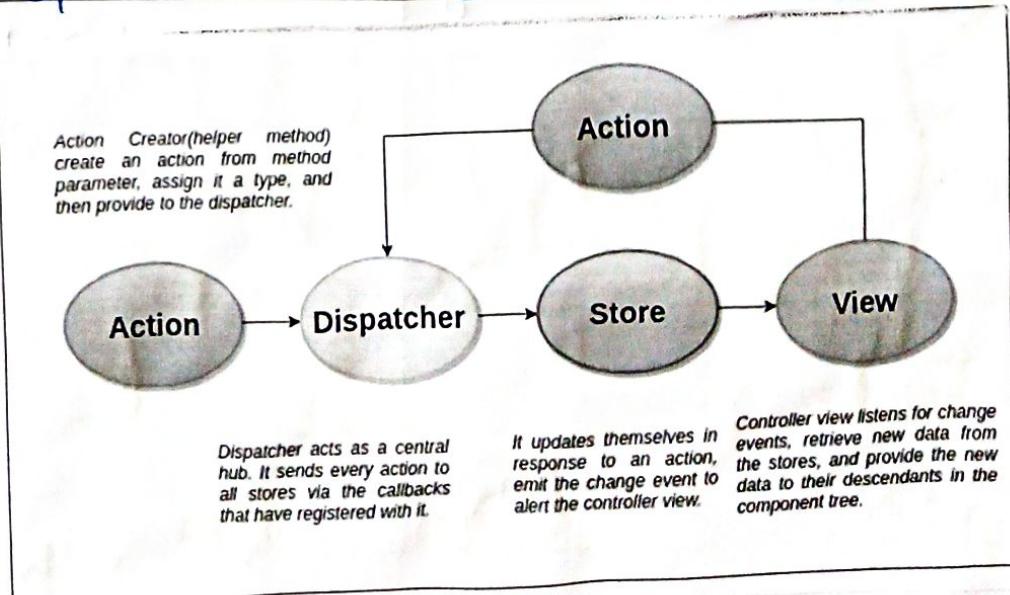
Click me



Flux :-

Flux is an application architecture that Facebook uses for building client-side web application. Flux architecture has 3 Major rules in dealing with data:-

- ① Dispatcher
- ② Store
- ③ views.



MVC

- ① It is introduced in 1976
- ② It supports Bi-directional data flow model
- ③ In this, data binding is the key
- ④ It is synchronous.
- ⑤ Controllers handle everything
- ⑥ It is hard to debug
- ⑦ It is difficult to understand as project size increases
- ⑧ Its maintainability is difficult as the project scope goes huge.
- ⑨ Testing of application is difficult.
- ⑩ Scalability is complex

FLUX

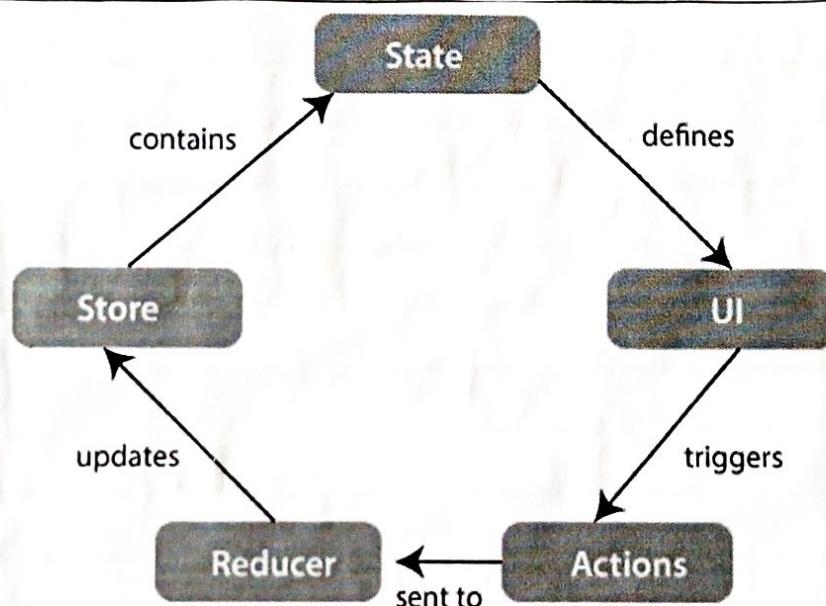
- It was introduced just a few years ago
- It supports Uni-directional data flow model
- In this events or actions are key.
- It is asynchronous.
- Stores handle all logic
- It is easy to debug.
- It is easy to understand.
- It maintainability is easy and reduces run-time errors.
- Testing of application is easy.
- It can be easily scalable.

React Redux:-

Redux was inspired by flux. Redux studied the Flux architecture.

- ① Redux does not have dispatcher concept.
- ② Redux has an only store whereas Flux has many stores.
- ③ The Action objects will be received and handled directly by Store.

React Architecture :-



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 function. It is like a brain responsible for all moving parts in Redux.

Action:- Action is sent or dispatched from the view has which are payloads that can be read by Reducers. It is pure object created

to the store the information of users even. It includes information such as type of action, time of occurrences, location of occurrences, its coordinates and which state it aims to change.

Reducer :- Reducer reads the payload from the action and then updates the store via the state accordingly. It is pure function to return a new state from the initial state.

React Portals :-

It is very tricky to render the child component outside of its parent component hierarchy. It breaks the convention when a component needs to render as a new element and follow a parent-child hierarchy.

`ReactDOM.createPortal(child, container)`

Example using Portal :-

We want to insert a child component in different location in the DOM

```
render() {
  return ReactDOM.createPortal(
    this.props.children,
    myNode,
  );
}
```

Features:-

- It uses React version 16 and its official API for creating portal.
- 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.
- It doesn't produce any DOM unness.
- It does no dependancies, minimalistics.

When to use?

① Modals.

② Tooltips

③ Floating menus.

④ Widgets

Explanation of React Portal

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;
```

```
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
```

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"/>
```

```
<title> React App </title>
</head>
<body>
<noscript> It required to enable Javascript
to run this app </noscript>
<div id = "root" > </div>
</body>
</html>
```

