# COMPLETE NOTES ON REACTJS

PREPARED BY

TOPPERWORLD

FOLLOW US

| | |
|---|---|
| WEBSITE | TOPPERWORLD.IN |
| LINKEDIN | TOPPERWORLD |
| INSTAGRAM | TOPPERWORLD.IN |

# INDEX

# Introduction to ReactJS

## What is ReactJS?

ReactJS is an opensource javascript library used for building user interfaces (UIs). It was developed for the view layer of the application by Facebook. React allows developers to create reusable UI components that efficiently update and render based on changes in data providing a responsive and interactive user experience.

## Advantages and Features of ReactJS:



**Virtual DOM**: React utilizes a virtual DOM (Document Object Model) that efficiently updates and renders changes to the UI. The virtual DOM manipulations, leading to improved performance and a smoother user experience

**Component-based architecture**: React promotes the concepts of reusable components, making it easier to build and maintain complex UIs. Components can be

composed togeather, allowing for code reusability and modularity

Declarative Syntax: React follows a declarative approach allowing developers to describe how UI should look based on its current state. This makes it easier to understand and reason about the UI

React Unidirectional data flow: React implements uni directional data flow, where data flows in single direction from parent components to child components. This makes it easier to track and debug data changes, reducing the likelihood of unexpected side effects.

React Native: React also provides react native, a framework that allows developers to build mobile applications using React and Javascript. with React Native, developers can create native mobile apps for both iOS and Android platforms using a single codebase.

Component allows developers to break down complex UI structures into smaller, manageable pieces, making the code easier to understand, maintain and reuse.

Components can be composed togeather to create larger and more sophisticated UI structures.

## React Components



### Class Components Vs Functional Components.

| Class Component | Functional Components |
|---|---|
| A class component requires you to extend from React.Component and create render function that returns React element. | A functional component is just plain Javascript pure function that accepts props as argument and returns React element (Jsx) |
| It must have render() returing Jsx | There is no render() used |
| Also known as State ful component | Also known as stateless component |

| | |
|---|---|
| React lifecycle methods can be used inside class components (Example: componentDidMount) | React lifecycle methods [Example: componentDidMount] cannot be used as Functional component |
| It requires different syntax inside a class component to implement hooks | Hooks can be easily used in functional components to make them stateful |
| Example<br>constructor (props){<br>super(props);<br>this.state = {name: ''}<br>} | Example<br>const [name, setName] =<br>React.useState(' ') |

# Creating and Rendering Components.

To create a React Component, we need to define either a class component or functional component.

Class Component Example:

```
import React from 'react'
class Welcome extends React.Component {
   render( ){
 return <h1> Hello, {this.props.name}! </h1>;
   }
}
export default Welcome;
```



Functional Component Example

```
import React from 'react'
Function Welcome(props) {
   return <h1> Hello, {props.name}! </h1>
   }
export default Welcome;
```

To render a component, we can import it into another component and include it in the JSx code using angle brackets

```
Function App(){
  return
  <div>
    <Welcome name = "Alice"/>
    <Welcome name = "Bob"/>
  </div>
  );
}
export default App;
```

In the above example, the 'Welcome' component is rendered twice, passing different names as props. This will result into two separate instance of 'Welcome' component being rendered with respective names.

## Setting up a development environment:

To set up a development environment for ReactJS, follow these steps

1. Install Node.js: React requires Node.js so begin by installing the latest version of Node.js from the official Node.js website.(http://node.org). Node.js comes with npm (Node package manager) which is used to manage project dependencies.



2. create a new React project: Once Node.js is installed you can create a new React project using the create React App (CCRA) command line tool. Open the terminal or command prompt and run the following.

```
npx create-react-app my-react-app
```

This will create a folder called my-react-app with basic structure and necessary files for React project

3. Navigate to project directory: Move into the project

by running the following command

```
cd my-react-app
```

4. Start the development server: To run the React development server, execute the following command.

```
npm start
```

This will start the development server and Launch your react Application in a web browser.

Any changes made to the code automatically trigger hot reload, allowing us to see the changes in real time.

## React Components

Understanding Components and their role in React

In React components are building blocks of user interface. It is a self-contained, reusable piece of code that represents a part of UI and encapsulates its logic, state and rendering.

# React Component life cycle

In ReactJs, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle.

The lifecycle of components is divided into 4 phases

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



## 1. Initial Phase

It is the birth phase of lifecycle of ReactJs component. component starts its journey on a way to DOM.

The initial phase occurs only once and consist of the following methods.

getDefaultProps( ) is used to specify the default value of this.prop. It is invoked before the creation of the component or any props from the parent is passed onto it

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

## 2. Mounting Phase

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

componentWillMount( )
This is invoked immediately before a components gets rendered into DOM. In this case, when you call setState( ) Inside this method, component will not re-render( )

componentDidMount( )
This is invoked immediately after the 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 single root html node element.

## 3. Updating Phase

The main aim of this phase is to ensure that component is displaying the latest version of itself. The phase consist of the following methods.

componentWillRecieveProps( )

It is invoked when a component recieves new props. if you want to update the state in response to prop changes, you should compare this.prop and nextProps to perform state transition by using this.setState( ) method



shouldComponentUpdate( )

It is invoked when component decides any changes/ updates to DOM. It allows you to control the component behaviour

ComponentWillUpdate( )

It is just invoked before the component updation occurs Here you can't change component state by invoking this.setState( ). It will be not called. if shouldCompo nentUpdate( ) returns false.

render()

It is invoked to examine this.props and this.state and returns one of the following types: React elements Arrays and fragments, Boolean or null, string and Number.

ComponentDidUpdate()

It is invoked immediately after the component updating occurs. In this method you can (any) code inside this once updating occurs

Unmouting Phase:

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

componentWillUnmount()

This method is invoked immediately before component is destroyed and unmounted.permanently.

It performs necessary cleanup related task.

If 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 : " Topperworld" };
      this.changestate = this.changestate.bind (this)
    }
  render( ) {
   return (
  <div>
   <h1> ReactJs components 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."});
}
componentWillRecievePropps (newProps ) {
  console.log("component will Recieve Props!");
}
shouldComponentUpdate(newProps, newState){
    return true;
  }
componentDidUpdate(prevProps, prevState){
   console.log('ComponentDid Update');
 }
componentWillUpdate(nextProps, nextState){
   console.log('component will update');
}
componentWillUnmount(){
  console.log('Component will unmount!')
 }
}
export default App;
```

Output

ReactJS component's Lifecycle

Hello Topperworld

click Here!

when you click on the button. you will get updated result shown below

ReactJS components Lifecycle

Hello All!! - Its a great reactjs tutorial

click Here!

©Topperworld

## ReactJSX

JSX (JavaScript Extension), is a React extension which allows writing javascript code that looks like html JSX is an HTML like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with Javascript / React code.
JSX allows you to write HTML/XML like structure in the same file where you write Javascript code.

Example

we will write Jsx syntax in Jsx file and see corresponding javascript code which transforms by preprocessor

Jsx File

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

corresponding Output

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

The above line creates react element & passing 3 arguments.

## Jsx Attributes:

Jsx use attributes with HTML elements same as regular HTML.

Example

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

```
<p data-demoAttribute = "demo" > This contains best
                                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 Java!
```



Example

```
import React { Component } from 'react';
class App extends Components {
 render( ) {
 return (
   <div>
   <h1 className = "hello" > Java</h1>
   <p data-demoAttribute = "demo" > This website
      contains topperworld info </p>
      </div>
   );
 }
}
export default App;
```

Output

Java

This website contains topperworld info.

## 2. As Expressions

we can specify the value of attributes as expressions using curly braces { }

var element = <h2 className = {varName} Hello

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 */ and wrapping them in curly braces { } just like in the case of Jsx expression.

## 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 no's values on specific elements.



## 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 overtime can happen as a response to user action or system event.

A component with state is known as stateful components.

It can set by using setstate(). It can only be accessed or modified inside the component.

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.

## Define State

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

Example:

Below shows how can we create 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>welcome</h3></p>
  </div>
    ):null;
    return(
      <div>
       <h1> Hello </h1>
    {bio}
    </div>
    );
  }
}
export default App;
```



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

Output

Hello
Welcome

# React Prop

## what are Props?

'Props' is short for 'properties' and they are a way to pass data from one React component to another. Props allow you to make your component dynamic and reusable by supplying different data to them.

## Declaring Props:

To use props, you define them in the component that recieves the data. Props are declared as parameters in the functional component or as properties in the class component

```
// Functional Component
function MyComponent(props) {
  return <div>{props.message}</div>
}
// class component
class MyComponent extends React.Component {
  render() {
    return <div>{this.props.message}</div>;
  }
}
```

In the above example. 'message' is a prop that can be passed to MyComponent

## Passing Props

To pass data to a component, you specify the prop as an attribute when using component.

```
<MyComponent message = "Hello, world" />
```

Here we are passing string "Hello, world" as 'message' prop to MyComponent



## Accessing Props

Inside recieving component you can access prop like a variable. In functional component, it is a parameter in class component it is accessed using this.prop

```
function Mycomponent (props) {
   return <div> {props. message } </div> ;
}
```

## Using Props Dynamically

Props can be used dynamically, allowing your components to display different data based on whats passed to them

Example

```
<MyComponent message="Hello world" />
<MyComponent message="welcome" />
```

## Default Props

you can set default values for props using 'defaultProps'. If a prop is not provided, default value will be used

Example

```
MyComponent.defaultProps = {
  message: "Default Message"
};
```

Props are fundamental concept in React that allows you to pass data from parent components to child components, making your application dynamic and allows you to create reusable components.

# 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 child component using props.

## Example

App.js

```
import React { Component } from 'react'
class App extends React.Component{
 constructor(props){
       this.state = {
          name: "TopperWorld"
       }
    }
 render( ) {
   return{
 <div>
 < JTP jtpProps = { 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> ReactJs handwritten notes from TopperWorld </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 TopperWorld

ReactJs handwritten notes from TopperWorld

# Difference between State and Props

| State | Props |
| --- | --- |
| state changes can be asynchronous | Props are read only |
| state is mutable | props are immutable |
| state cannot be accessed by child components | Props can be accessed by child components. |
| stateless component cannot have state | stateless components can have props |
| state cannot make component reusable | Props can make component reusable |
| The state is inter -nal and controlled by React component itself | Props are external and controlled by whatever render the component. |
| state contain default values | Props also contain default values. |

# React Constructor

## what is a constructor?

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

syntax

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

In React, constructors are mainly used for two purpose

- Initialising
- binding event handlers

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

The constructor uses this.state to assign initial values and all other methods needs to use set.State()

Example

App.js

```
import React, { Component } from 'react';
class App extends Components{
  constructor (props){
    super(props){
     this.state ={
       data: Topperworld.in
     }
    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

React Constructor Example

| toperworld.in | Please click |
|---|---|

## React Fragments

When you want to render something, you need to use render() inside component

This render method can return single or multiple elements.

If you want to return multiple elements. The render method requires 'div' tag and put entire content inside it

This extra node to DOM sometimes results in wrong formatting of your HTML output.

Example

```
//Rendering with div tag
class App extends React.component {
  render() {
    return (
      // div element
      <div>
        <h2> Hello world </h2>
      </div>
    );
  }
}
```

© Topperworld

To solve the above problem React introduced fragments Fragments allow you to group list of children without adding extra nodes to DOM

Syntax

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

## Example

```
class App extends React.Component {
 render() {
  return (
    <React.Fragment>
        <h2> Hello World </h2>
     <p> welcome to TopperWorld </p>
    </React.Fragment>
  );
 }
}
```

## why we use Fragments

The main reason to use fragments is

It makes execution of code faster as compared to div tag

It takes less memory

## Fragment short Syntax

There is also another shorthand exists for declaring fragments. It looks like empty tag in which we can use '< >' and " Instead of 'React.Fragment'.

## Example

```
class Columns extends React.Component{
  render(){
    return(
    <>
     <h2> Hello          </h2>
     <p> welcome to Topperworld </p>
    </>
    );
  }
}
```



## Keyed Fragments

The shorthand syntax does not accept key attributes key is the only attribute that can be passed with the fragments.

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 fragments with explicit syntax

```
Function = (props) {
return (
<Fragment>
  { props.items.data.map (item => (
  //without 'key' React will give warning
  <React.Fragment key = {item.id} >
    <h2> {item.name} </h2>
    <p> {item.url} </p>
    <p> {item.description} </p>
    </React.Fragment>
  )) }
</Fragment>
)
}
```

# 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

creating elements
Transforming element
Fragments

## setState()

This method is used to update state of component
The method does not always replace the state immediately. It only add changes to original state
It is a primary method that is used to update user interface in response event handler and server response



syntax

```
this.state.state(Object newstate [function. callback]);
```

```
import React, {Component} from 'react';
import PropTypes from 'prop-types';
class App extends React Component{
constructor(){
```

```
super( ){
this . state ={
msg = "welcome"
};
this .update .setState = this .updatesetState .bind (this);
}
updatesetState ( ) {
    this . setstate( {
        msg . " Its best tutorial"
    });
render ( ){
  return (
   <div>
  <h1> { this .state .msg } </h1>
<button onClick = {this .updatestate } >
  </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'));
```

Output

Welcome

SETSTATE

when you click on the setstate button, you will see the following shown below

Its the best ReactJS tutorial

SETSTATE

ForceUpdate()

This method allows you 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 : 0.5639401820807 52

ForceUpdate

each time when you click on force update button you will see the following below

Example to generate random number

Random number : 0.31739060740251275

ForceUpdate

## findDOMNode()

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

syntax

```
ReactDOM.findDomNode(component);
```

Example

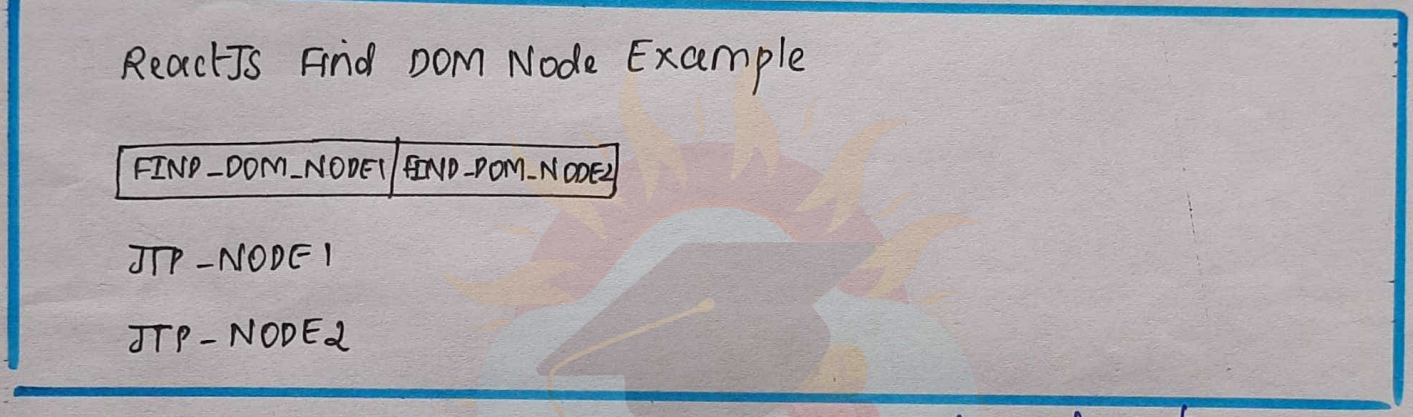
App.Js

```
import React {Component} from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component{
constructor(){
    super();
```

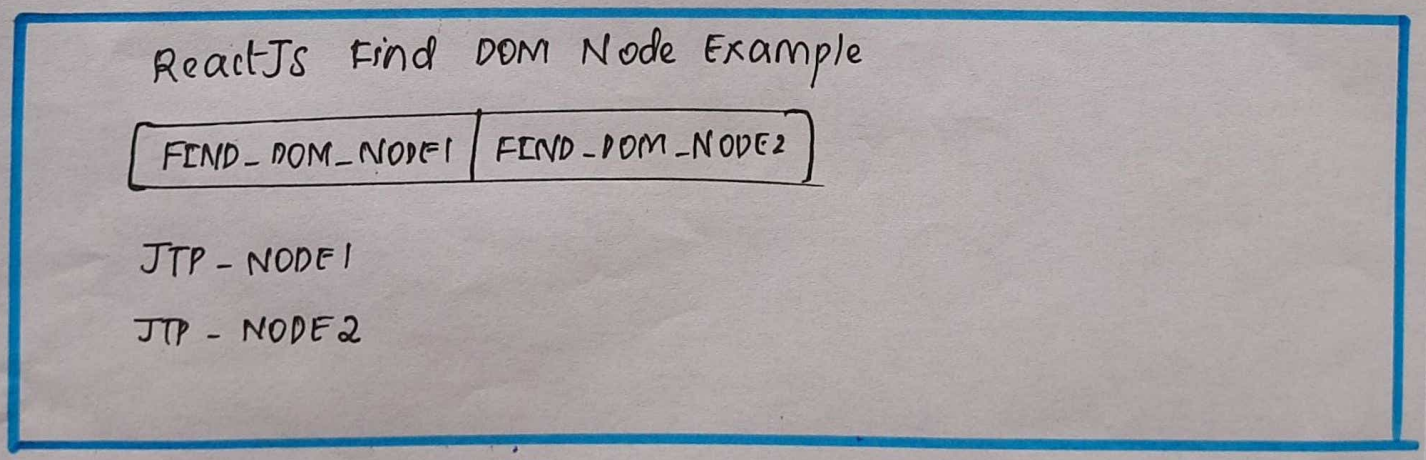
```
this.findDOMNodeHandler1 = this.findDOMNode.Handler1.
                           bind(this);
this.FindDomNodeHandler2 = this.findDOMNode.Handler2.
                           bind(this);

};

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_NODE1 </button>
<button onclick ={this.FindDomNode Handler2}>
          FIND_DOM_NODE 2 </button>
<h3 id = "myDivOne"> JTP_Node1 </h3>
```

```
<button onClick={this.forceUpdate.state}>
        Force Update</button>
  </div>
  );
 } }
 export default App;
```

Output




once you click the button, the color of node gets changed.



# React Forms

Forms are an Internal part of any modern web application It allows user to interact with applications as well as gather information from the user. Forms perform many task that depends on nature of your business requirements and logic such as authentication of user

## Creating Forms

React offers a stateful, reactive approach to build a form. The components rather than DOM usally handles the React form

1. uncontrolled Component
2. Controlled Component



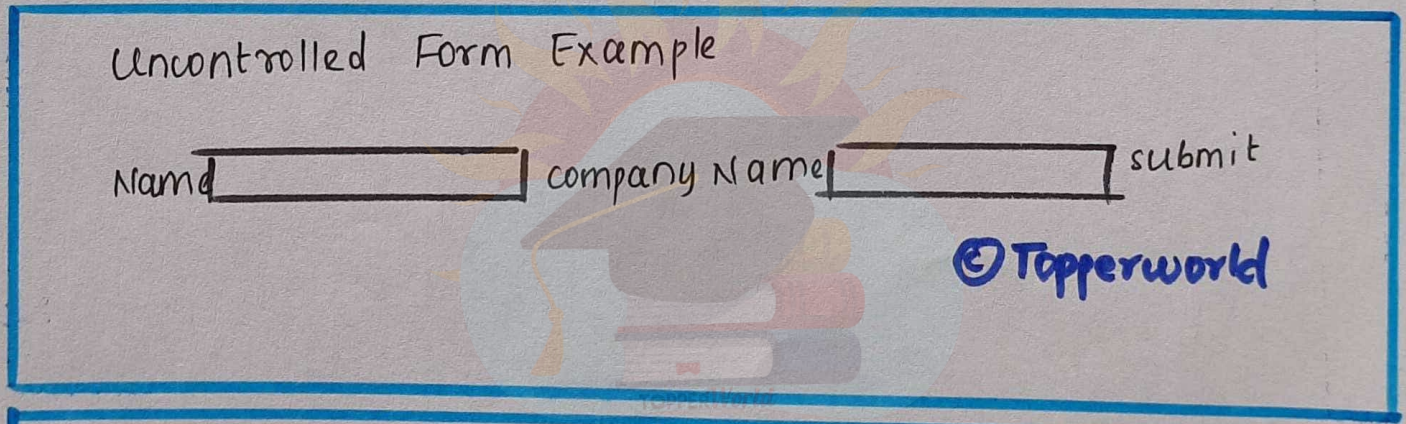
## Uncontrolled Component

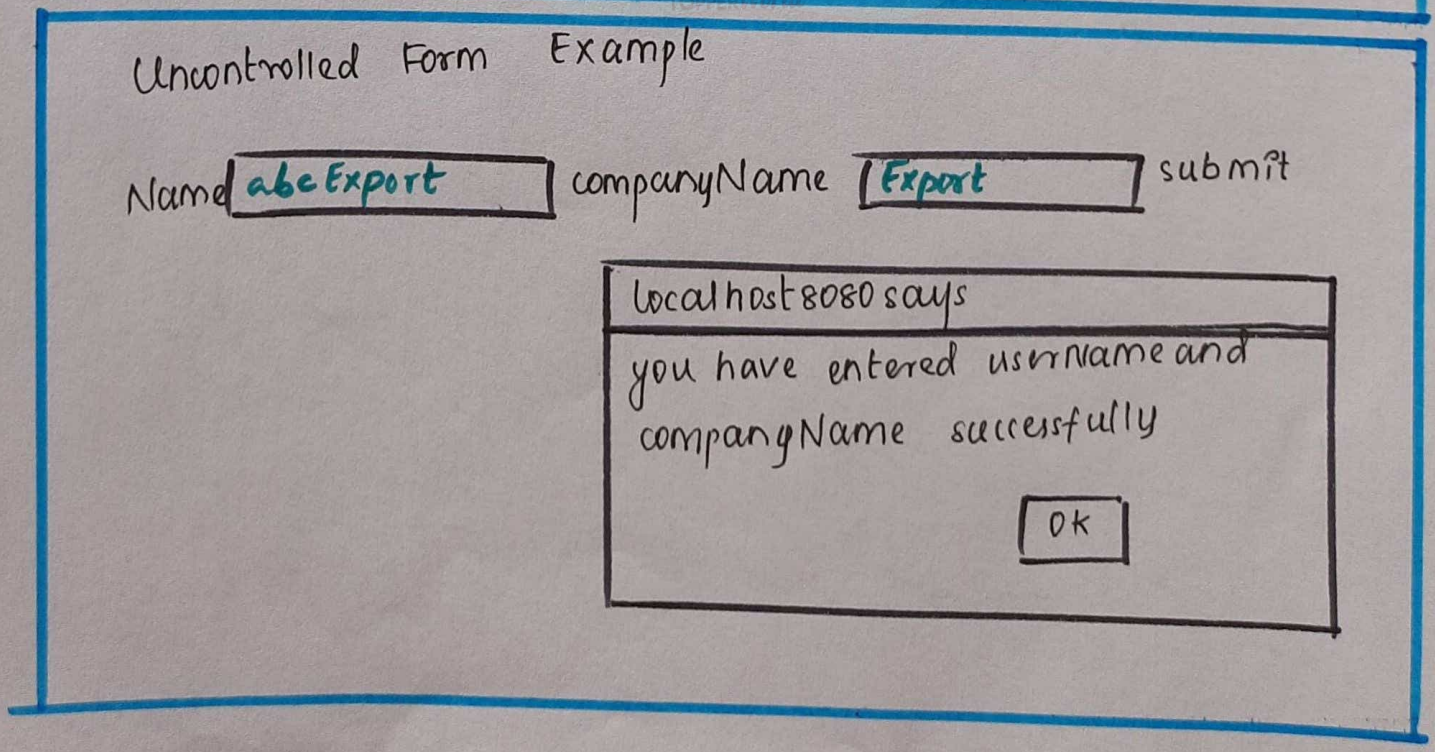
The uncontrolled component input is similarto the traditional HTML form input. The DOM itself handles the form data. The Html element maintain their own state that will be updated when Input value changes To write an uncontrolled component.

```
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
          company name successfully");
    event.preventDefault();
    }
 render() {
   return ( )
  < form  onSubmit = {this.updateSubmit} >
    <h1> Uncontrolled form </h1>
   < label > Name:
<input type = "text" ref = {this.input} />
 </Label>
    <label>
    Company Name :
```

```
<input type = "text" ref = {this.input}/>
        </label>
      </form>
    );
  }
}
export default App;
```

Output



# Controlled Component

controlled components have function that goes on data passing into them on every onchange event, rather than grabbing data only once

```
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( )
{
 returns (
 <form onSubmit ={this.handleSubmit}>
 <h1> Controlled form </h1>
 <label>
  Name:
 <input type ="text" value ={this.state.value}
  onChange ={this.handleChanges }/>
    </Label>
 <input type ="submit" value ="submit" />
  </form>
  );
 }
}
export default App;
```

# Controlled Component Vs Uncontrolled Component

| Controlled | Uncontrolled |
| --- | --- |
| It accepts current value as prop | It uses ref for current values |
| It has better control over form elements and data | It has limited control over form elements |
| It does not maintain internal state | It maintains internal state |
| Data is controlled by parent component | Data is controlled by DOM |
| It allows validation control | It does not allow validation control. |

# React Events

React has same events as HTML : click, change, mouseover etc

## Adding events

React events are written in camelCase syntax

onClick instead onclick

React event handler are written inside curly braces

onClick ={ shoot } instead of

onclick = " shoot ( ) "



React

```
<button onClick={shoot}>Take shot!</button>
```

HTML

```
<button onclick="shoot()">Takeshot!</button>
```



## Passing Arguments

To pass an argument to event handler use an arrow function

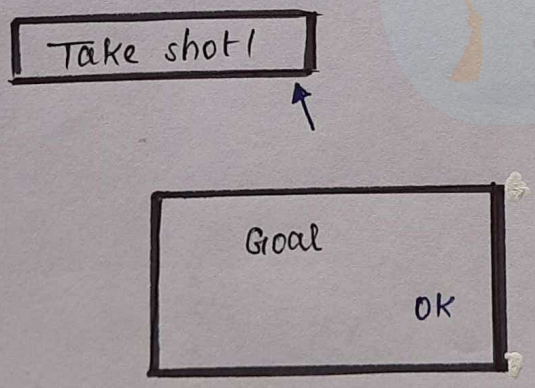
### Example

Send "Goal" as parameter to shoot function

```
function Football (){
  const  shoot  = (a) => {
   alert (a);
  }
  return (
     <button onClick = { ( ) => shoot (" Goal ") } > Take shot!
     </button>
     );
  }
const root  = ReactDOM. createRoot (
root. render ( <Football/> ) ;
```



output



# React Conditional Rendering

## If statement

Example

```
function MissedGoal( ) {
  return <h1> MISSED! </h1> ;
}
```

## Logical && Operator

we can embed javascript expressions in Jsx by using curly braces.

```
function Garage (props) {
const cars = props.cars ;
return (
<>
  <h1> Garage </h1>
  {cars.length > 0 &&
    <h2>
      you have {cars.length} cars in your garage.
      </h2>
  }
</>
);
}
```

```
const cars = ['Ford', 'BMW', 'Audi'];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage cars={cars} />);
```

Output

```
Garage
You have 3 cars in your garage
```

Ternary Operator

```
condition ? true : false
```



Example

```
function Goal(props) {
 const isGoal = props.isGoal;
 return (
 <>
  {isGoal ? <MadeGoal /> : <MissedGoal />}
 </>
 );
}
```

```
const root = ReactDOM.createRoot(document.getElementById
                                  ('root'));

root.render(<Goal isGoal = {false}/>);
```

output

```
MISSED
```

## React List

list are used to display data in ordered format & mainly used to display menus on websites

Example

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

output

```
[5, 10, 15, 20, 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 list.

### Example

```
const stringLists = ['Peter', 'sachin', 'Kelvin', 'Dhoni']
const updatedLists = stringList.map((strList) => {
<li key={strList.id}>{strList}</li>;
});
```



## React Refs

Refs is shorthand used for reference. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements.

### When to use Refs:

When we need DOM measurements such as managing focus, text selection or media playback.

It can also use as in call back

## When not to use Refs

Instead of using open( ) and close( ) methods on a Dialog components, you need to pass an isOpen prop it

You should avoid overuse of Refs

## React Router

Add react router in your application, run this in terminal from root directory of application

```
npm i -D react-router-dom
```



### Basic usage

Now use router index.js file

### Example

```
export default function App() {
return (
  <BrowserRouter>
    <Route path="/" element={<Layout/>}>
    <Route index element={<Home/>}/>
    <Route path="blogs" element={<Blogs/>}/>
    <Route path="contact" element={<Contact/>}/>
    <Route path="*" element={<NoPage/>}/>
  </Route>
  </Routes>
```

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

**Output**

localhost:3000

Home
Blogs
Contact

---

**Blog Articles**



## Benefits of ReactRouter

It is not necessary to set browser history manually

Link uses a navigate internal links in application.

It uses switch features for rendering

If the router need only single child element

# Styling using Css (React)

There are many ways to style React with CSS.

Common ways are

- Inline styling
- CSS stylesheets
- CSS Modules

## Inline styling

To style an element with Inline style attribute.

value must be javaScript Object



Example

```
const Header = ( ) =>{
  return(
   <>
   <h1 style = {{color: "red"}}>Hello Topper </h1>
   <p> welcome to Topperworld! <p>
  </>
  );
}
const root = ReactDOM.createRoot(document.getElementById
                                  ('root'));
root.render (<Header/>);
```

Output

Hello Topper

welcome to Topperworld!

Note:

In JSX, Javascript expression are written inside curly braces, and since javascript objects also uses curly braces, styling above written inside two set of curly braces.

Camel Case Property Names.

Example

use backgroundColor instead of background-color:

```
const Header = () => {
return (
   <>
   <h1 style = {{ backgroundColor : "lightblue" }}>
            Hello </h1>
   <p> Add style </p>
  </>
  );
}
```

```
const root = ReactDOM.createRoot(document.getElementById
                                  ('root'));

root.render(<Header/>);
```

Output

**Hello**

Add style

Javascript object

you can also create object with styling

©Topperworld

Example

~~create style object named myStyle:~~

```
const Headur = ( ) => {
    const myStyle = {
        color : "white"
        font-family :"sans-serif"
        backgroundColor : "red"
        };
    return (
        <>
```

```
<h1 style = {myStyle}> Hello </h1>
  <p> welcome to React </p>
  </>
  );
}

const root = ReactDOM.createRoot(document.getElementById
                                  ('root'));

root.render(<Header />);
```

Output

HELLO

welcome to React

## CSS Stylesheet

write your css styling in .css file extension and import it in your application

create App.css and insert some css code inside it

App.css

```
body {

background-color : #282c34;
color : white;
font-family : sans-serif ;

}
```

## Css Modules

Another way of adding styles to your application is using css modules

Css modules are convenient for components that are placed in separate files.



create css module with .module.css extension

example my-style.module.css

my-style.module.css

```
.bigblue{

color : DogerBlue;
font-family : sans-serif;
text-align : center;
padding : 40px;

}
```

# React Redux

React Redux is a predictable state container for javascript application

It helps you write apps that behave consistently run in different environments (client, server, native) and are easy to test.

Redux is a state management tool

Redux can be used with any Javascript framework

Redux stores the state of application, and the components can access the state from state store

## Principle of Redux

The three most important redux principles are

Single source of Truth

State is Read only

changes are made with pure functions

## Single Source of Truth

The state of your whole application is stored in an object tree within single-store

A single state tree makes it easier to debug or inspect an application

It gives you faster development of cycle by enabling you to persist in you app's navigation state

## State is read only

The only way to change the state to initiate an action an object describing



This feature ensures that no events like network callback or views can change state.

Actions are just plain objects, they can be logged.

## changes are made with pure functions.

To specify how actions transform state tree.

The user can return new state objects instead of mutating previous state

## Pillars of Redux

These are redux main pillars:

store

A store is an object that holds applications state tree

getstate() - returns current state

dispatch() - dispatches actoin

subscribe() - change listener to state

unsubscribe() - no longer call your listner method when state changes

## ReactJS Portals

React portals come up with a way to render children into DOM node that occurs outside DOM hierarchy of parent component.

### syntax

The createPortal ( ) method from ReactDOM is used to create portal in react. The method has the following syntax

ReactDOM.createPortal(child, container)

The child is a react component of fragment that can be rendered in DOM

The container is the reference to the element that is present in HTML page to which child element will be rendered

## Features:

React portals can be used to render a child component

Flexible component rendering

Improved performance

Integration with third party library

Encapsulation and isolation

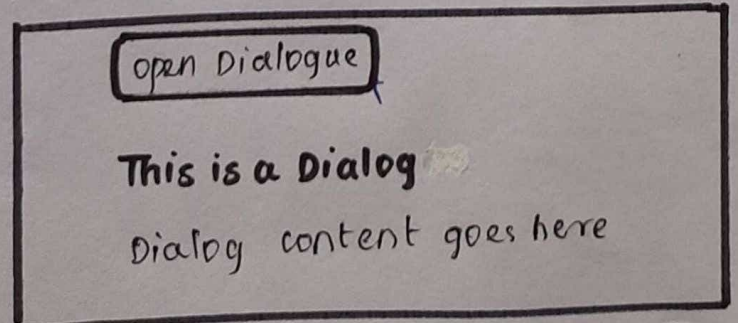


## Dialog

Dialog are modal components that overlay the main content to display important messages, notification or user interaction

```
import React from 'react';
import ReactDOM from 'react-dom';
const DialogExample = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;

  return ReactDOM.createPortal(
  <div className="dialog-overlay">
   <div className="dialog-content">
     <button className="close-button" onClick={onClose}>
      Close
     </button>
     {children}
     </div>
  </div>,
  document.getElementById('dialog-root')
  );
};
export default DialogExample;
```

output




when you click open dialog button "This is a Dialog Dialog content goes here" will be displayed.

# React Best Practices and Performance Optimization

## Best Practices for organising and structuring React

### Component Organisation

Organise the components into logical and reusable units use a component-based architecture, where each component has single responsibility and can be easily reused in different parts of application.



### File Structure

Follow a consistent file structure to keep our database organised. consider grouping related files togeather such as components, styles and tests. to improve maintainability and ease of navigation.

### Container and Persentational Components.

Implement the container-presentational component pattern. Container components are responsible for managing state, handling data, fetching and

interacting with redux store. Presentational components focus on rendering UI elements. and recieving props to display data

## Separate concerns

Keep concerns separated between components. separate rendering logic from business logic, and avoiding mixing presentational skills and management code in same component

## Use React hooks

Utilize React hooks, such as useState, useEffect and useContext to manage component state, handle side-effects and share state across components. Hooks provide a more concise and read-able way to write functional components.

## Maintain Readability

write clean, readable code by following naming conventions using descriptive variable and function names and commenting when necessary. Break down complex components to smaller, more managable components.

# Performance Optimization Techniques in React

## Memoization

use the React.memo higher order component or the use Memo hook to memoize and cache the results of expensive calculations or component renderings. This can prevent unnecessary re-renders and improve performance.

## virtualization

Implement virtualization technique for large list or tables to render only visible pattern. rather than rendering the entire list. libraries like react-virtualised and react window can help with virtualizing large data sets.



## Code splitting

split the code into smaller chunks and load them on demand using techniques like dynamic import or React. lazy for Lazy Loding. This can reduce the initial bundle size and improve the application loading performance.

## Performance Profiling:

Use tools like React profile, React Dev Tools, or browser DevTools to identify performance bottlenecks in the application. Measure and optimize components or operations that cause performance issue, such as excess re-renders or inefficient data fetching

## Debugging and Troubleshooting common issues

### Use developer Tools

leverage the browser developer's such as Javascript console, ReactDev Tools to debug and inspect our react application.

These tools provide insights into component hierarchies state changes, and potential errors.

### Debugging Techniques

Implementing console logging or use debugger statement

to examine variable values, track components. lifecycle, or identify unexpected behaviour. This can help pinpoint the source of issues and assist in debugging

## Error Handling

Implementing pooper error handling techniques such as using try catch blocks, error boundaries or global error handling components like ComponentDidC -atch

This ensures that errors are caught and gracefully handled, preventing application crashes or unexpected behaviour.



## Testing

write unit test for react components and use linting tools like Eslint to catch common coding errors. and enforce best practices. Testing and linting can help identify issues early in development process.

# REACTJS INTERVIEW QUESTIONS

1. which of the following is used to React.js to increase performance?

a. Virtual DOM
b. Original DOM
c. Both A and B
d. Non of the above

2. what is ReactJs?

a. Server side Framework
b. Both a and b
c. none of the above
d. user Interface framework



3. Identify one which is used to pass data to components from outside

a. Render with arguments
b. setState
c. propTypes
d. props

4. who created Reactjs?

a. John Mike
b. Time Lee
c. Jordan Walke

5. In which Language is React.js written?

a. python

b. Javascript

c. Java

d. PHP

6. what is Babel?

a. Javascript compiler

b. Javascript Interpreter

c. Javascript transpiler

d. none

7. Identify the command used to create react app

a. npm install create -react- app

b. npm install -g create -react- app

c. install -g create -react- app

d. none



8. How many components can a valid react component return?

a 1

b 2

c 3

d 4

9. A state in React.js is also known as?

a. **The internal storage of component**

b. external storage of component

c. permanent storage

d. all of the above

10. state whether true or false

Props are methods into other component

a. **True**

b. False

11. what are arbitary inputs of components in react also known as?



a. Elements

b. **Props**

c. Key

d. Refs

12. State whether true or false? React.js cover only view layer of app

a. **True**

b. False

13 which of the following is not a part of ReactDOM

a. ReactDOM.hydrate()

b. **ReactDOM.destroy()**

c. ReactDOM.createPortal()

d. All

14 In which of the following directory react components are saved?

**a. Inside js/components/**

b. Inside components/js

c. inside vendor/js/components

d. inside vendor/components.

15 JSX stands for

a. Javascript Extention

b. Javascript

c. **Javascript xml**

d. None

16 Why JSX used for

a. **To write HTML in react**

b. To write jquery in React

c. None

d. To write SQL

17. what do you mean ES6?

a. Javascript 6

b. Extended Javascript 6

c. **ECMAScript 6**

d. None

18. what is the purpose of useEffect hook in React

a. To add events

b. To render components

c. To perform and manage state

d. **To perform side effects**

19. what is Redux store



a. **A single state tree**

b. Extended Javascript 6

c. None

d. To render redux

20. why react is mainly used for

a. Database

b. MVC

c. Extention

d. **User interface**