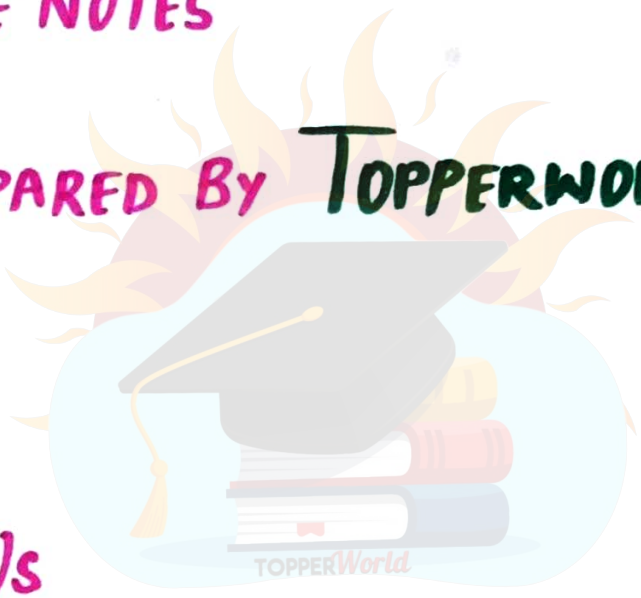
# REACT JS

COMPLETE NOTES

PREPARED BY TOPPERWORLD




FOLLOW US

WEBSITE TOPPERWORLD.IN

LINKEDIN TOPPERWORLD

INSTAGRAM TOPPERWORLD.IN

# INDEX

# REACT JS



## What is ReactJS?

ReactJs is an open source Javascript library used for building interfaces. It focuses on creating dynamic interactive web applications by allowing developers to create reusable UI components.

## Why use ReactJS?

ReactJs is popular and widely used for several reasons:

### Component Based Architecture

React allows you to break down your UI into reusable components. making it easier to manage.

### Efficient Updates:

It uses virtual DOM, which optimizes rendering by updating only parts of pages that change improving performance

### Large Ecosystem:

React has vast ecosystem of libraries and tools

### Cross platform:

React can be used for both web and mobile app development.

# PROS AND CONS OF REACTJS

## PROS:

COMPONENT-Based React encourages a modular, component based approach to build user interfaces.

Easy to learn and use: React is much easier to learn and use. It comes with good supply of documentation

Reusable components: A ReactJS web application is made up of multiple components. each component has its own logic

Scope for Testing Codes: ReactJS applications are extremely easy to test. It offers scope where developers can test and debug their codes with help of native tool

## CONS:

Poor Documentation React Technologies updating and accelerating so fast that there is no time to make proper documentation

view part ReactJs covers only VI layers of app & nothing else. so you still need to choose other technologies to get complete tooling set for development

# DIFFERENCE BETWEEN REACTJS & ANGULAR

©Topperworld

| Field | React js | Angular |
|---|---|---|
| used as | Reactjs is a library (Javascript) | Angular is a framework |
| Released | It was released in 2013 | It was released in 2010 |
| Written | Reactjs written in javascript | Angular is written in microsoft's Typescript |
| Routing | Routing is not easy in ReactJS | Routing is easy compared to ReactJS |
| Scalable | It is highly scalable | It is less scalable |
| DOM | It has a virtual DOM | It has regular DOM |
| Testing | It supports unit Testing | It supports unit & integration testing |
| Data Binding | It supports uni directional | It supports bi directional |

# Difference between ReactJS and React Native

| SN | ReactJS | ReactNative |
|---|---|---|
| 1 | Installation process<br>ReactJs library is installed via npm package manager | React Native is a command line interface tool requires both Node.js and React Native CLI to be installed. |
| 2 | Efficiency<br>ReactJS is more efficient interms of code reuse ability | React Native is more efficient interms of performance & memory usage |
| 3 | Technology Base<br>ReactJS is javascript library used for building user interfaces | React Native is a cross platform mobile development |
| 4 | Components<br>ReactJs components are typically written in HTML | ReactNative components are written in JSX. |
| 5 | Navigation<br>ReactJs uses traditional browser based approach | ReactNative relies on native platform |

| ReactJs | ReactNative |
| --- | --- |
| Storage<br>ReactJs is a good choice for projects that require high performance | It is a good choice for projects that need to be able to scale easy |
| Search engine friendly<br>It is more search engine friendly than ReactNative | It cannot be made search engine friendly |
| Platform:<br>React is a framework for building application using javascript | It allows building native and cross platform mobile apps |
| Rendering<br>Browser code is rendered through virtual DOM | It uses native API to render all components |
| Syntax<br>Makes use of HTML and its syntax flow | ReactNative uses ReactNative syntax |

## Key difference between React and Vue.

The main distinction between vue and React is how they approach application design

While React focuses on creating reusable UI components vue takes approach by providing developers with frontend tools.

Let now take a look at some particular are differences

| Vue | React |
| --- | --- |
| It uses single file components (SFC) to build different components | It uses JSX as a component format |
| It is used to dev -elope web-based application | It is used do develop web as well as mobile application |
| State Management library is called Vuex | State Management library is called Redux |

| | |
|---|---|
| The performance is slow in react<br><br>It is not suitable for long term support | The performance is slow when compared to Vue<br><br>It is suitable for long term support |

## React JSX

what is JSX?

JSX stands for javascript XML

JSX allows us to write HTML in React

JSX makes it easier to write and add HTML in react



### Coding JSX

JSX allows us to write HTML elements in Javascript and place them in DOM without creating createElement() or appendChild()

JSX converts HTML tags into react elements.

you are not required to use JSX, but JSX makes it easier to write React application

## Example 1

JSX:

```
const myElement = <h1> I love JSX! </h1>;
const root = ReactDOM.createRoot(document.getElement
                                  ById('root'));

root.render(myElement);
```

## Output

I Love JSX!

## Example 2

without JSX:

```
const myElement = React.createElement('h1', {}, 'I do not');
const root = ReactDOM.createRoot(document.getElementById
                                  ('root'));
root.render(myElement);
```

## Output

I do not

In example 1. JSX allows you to write HTML directly within Javascript.
JSX is an extension of Javascript language based on ES6.

## Expression in JSX

With JSX you can write expression inside curly braces { }
The expression can be React variable or property. JSX will execute expression and return the result.

### Example

execute the expression 5 + 5

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```

### Output

React is 10 times better with JSX



### Inserting a large block of HTML

TO write HTML on multiple lines, put HTML inside parenthesis

### Example

```
const myElem = (

  <ul>
    <li>Apple </li>
    <li>Banana</li>
    <li>cherries</li>
  </ul>
);
```

**Output**

- Apple
- Banana
- cherries

**Example**

```
const myElement = (
  <div>                =  (
    <p> I am paragraph </p>
    <p> paragraph too </p>
  </div>
  );
```

⇐ JSX will throw error if HTML is not correct

or if HTML misses a parent element

Here  is parent <p> is child

**Output**

I am paragraph

paragraph too

Elements must be closed in

close empty elements with />

## Setting up a React Environment

If you have npx and node.js installed, you can create a React application by using create-react-app

The create-react-app will set up everything you need to run a React application

Run the React Application

cd my-react-app

Run this command to run react application

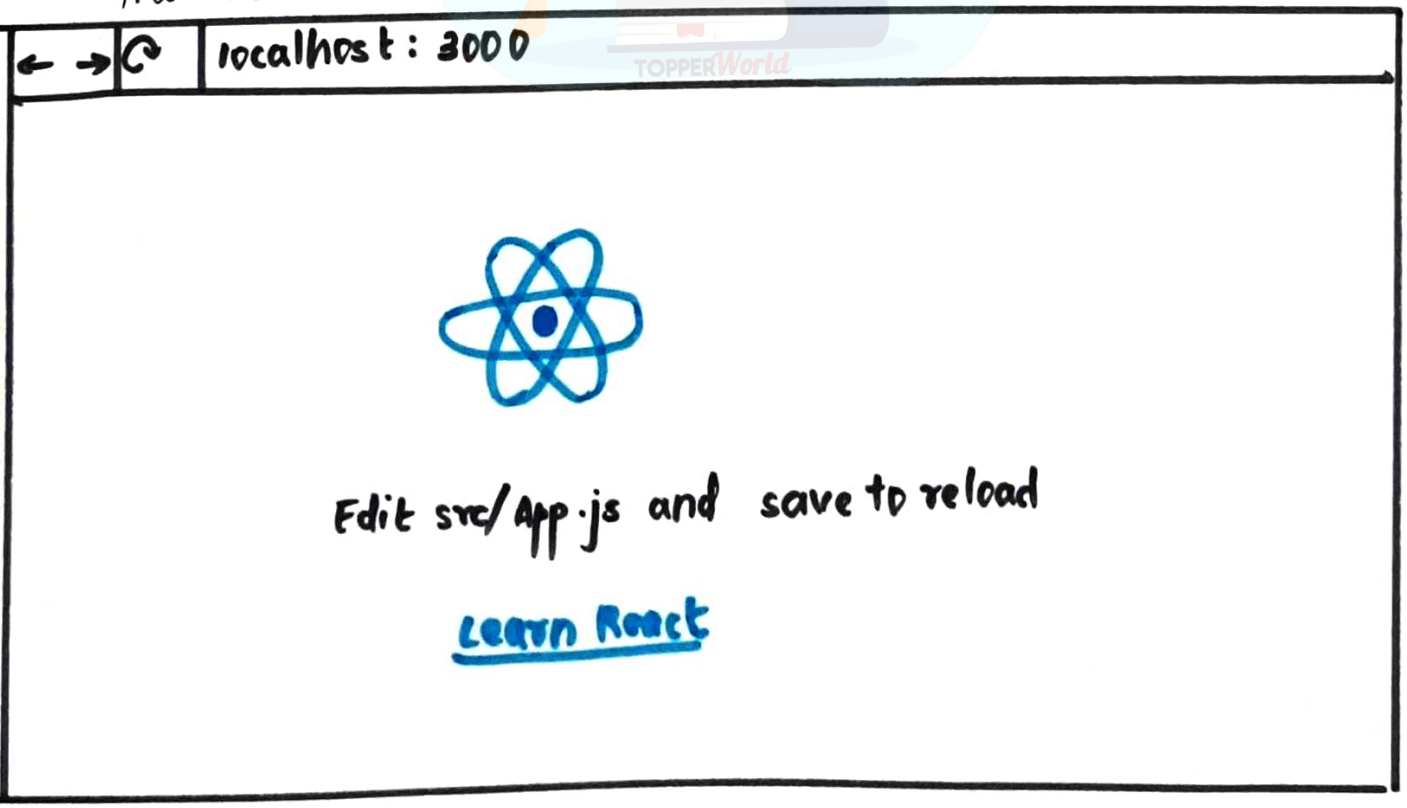


npm start

A new browser will pop up with your newly created ReactApp!.

If not open browser and type localhost:3000

The result:



## React Components

When creating a React Component, the component name must start with Uppercase. letter.

Class Component must include extends. React. Component statement

The component also requires render(). this method returns HTML

**Example**

```
class Car extends React.Component {
  render() {
    return <h2> Hi, I am a car! </h2>
  }
}
```

**Function component**

Same as react component, only difference is written using less code

```
function car() {
  return <h2> Hi, I am a car! </h2>;
}
```

**Props:**

Components can be passed as props, which stands for properties.

## Component in Files

React is all about re-using code and it is recommended to split your components into separate files.

To do that create a new file with .js file

## React class Component state

React class components have built-in state object

## Creating the State Object

```
class Car extends React.Components {
  constructor (props) {
  super (props);
this.state = {brand: "ford"};
  }
  render(){
    return (
    <div>
      <h1> My Car </h1>
    </div>
    );
  }
}
```

## Using the state Object

Refer to the state object anywhere in the component using this.state.propertyname syntax

Example

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford"
      model: "Mustang"
      color: "red"
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}
        </p>
      </div>
    );
  }
}
```

Output

My Ford

It is a red Mustang from 1964

## Changing the State Object

To change a value in state object, use the this.setState() method.

Example:

```
class Car extends React.Component {
constructor(props)
super(props);
this.state = {
    brand = "Ford",
    model = "Mustang",
    color = "red",
    year = 1960.
    };
}
changeColor = ( ) => {
     this.setState({color: "blue"});
}
render( ){
  return(
```

```
<div>
  <h1> My {this.state.      }</h1>
  <p>
    It is a {this.state.color}
    {this.state.model}
    from {this.state.year}
  </p>

  <button
      type = "button"
      onClick = {this.changeColor}
      >change color</button>
  </div>
);
}
}
```

Output

**My Ford**

It is a red Mustang from 1964

change color (when you click)

**My Ford**

It is a blue Mustang from 1960

change color

# Lifecycle of Components

Each component in React has a lifecycle which you can monitor and manipulate during three main phases.

The three phases are

- Mounting
- Updating
- Unmounting

## Mounting

Mounting means putting elements in DOM.

React has four built in methods that gets called, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

## 1. Constructor

The constructor method is called by React, every time you make a component

Example

```
class Header extends React.Component {
  constructor(props) {
  super(props);
  this.state = {favouritecolor: "red"};
  }
  render() {
  return {
  <h1>My favourite color is {this.state.favouritecolor}</h1>
  );
  }
}
  const root = ReactDOM.createRoot(document.getElement
                                    ById('root'));
  root.render(<Header/>);
```



**output**

My Favourite Color is red

2. getDerivedStateFromProps

The getDerivedStateFromProps() is called right before element(s) in the DOM

The example below starts with favourite color being "red", but the getDerivedStateFromProps() updates the favourite color based on favcol attribute

Example

```
class Header extends React.Component {
  constructor(props) {
  super(props};
  this.state = {favouritecolor:"red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favouritecolor: props.favcol};
  }
  render() {
    return(
      <h1>My Favourite Color is {this.state.favourite
                                  color</h1>
    );
  }
}
```

```
const root = ReactDOM.createRoot(document.getElement
                                  ById('root'));
root.render(<Heady favcol="yellow"/>);
```

output

My Favourite Color is yellow

## Render

render() is required and is the method that actually outputs HTML to the DOM

Example

```
class Header extends React.Component {
render() {
 return (
   <h1>This is the component of Header</h1>
  );
 }
}
const root = ReactDOM.createRoot(document.getElement
                                  ById('root'));
root.render(<Header/>);
```

Output

This is the component of Header

## 4. componentDidMount

componentDidMount() method is called after the component is rendered

Example

At first my favourite color is red, but give me a second, and it is yellow instead



```
class Header extends React.Component {

constructor(props){
    super(props);
    this.state={favouritecolor:"red"};
    }

componentDidMount(){
    setTimeout(()=>{
    this.setstate({favouritecolor:"yellow"})
    },1000)
}
```

```
render() {
  return (
   <h1> My favourite color is {this.state.favouritecolor}</h1>
  );
 }
}

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

Output

My favourite Color is yellow

Updating

The next phase in lifecycle is when component is updated.

React has five built in methods that gets called when component is updated

# React Props

props are arguments passed into React components.

props stands for properties

## Example

Add a brand name attribute to car element

```
const myElement = <Car brand = "Ford" />;
```

## Example

use brand attribute in the example



```
function Car (props)
  return <h2> I am a {props.brand}! </h2>;

const myElement = <Car brand = "Ford" />;

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

## Output

I am a Ford!

# 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>
```

## Pasing Arguments

To pass an argument to event handler use an arrow function

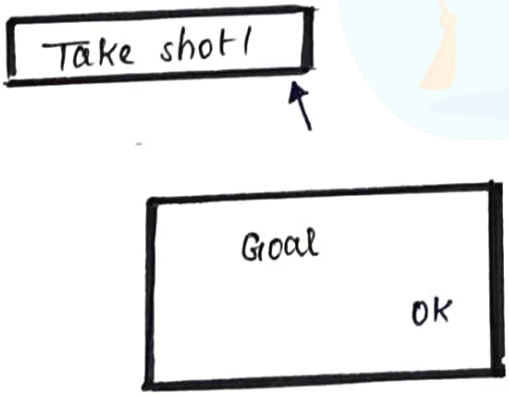
### Example

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

## How to access Refs

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

## Callback refs

Another way of using refs is callback ref and it gives more control when refs are set and unset

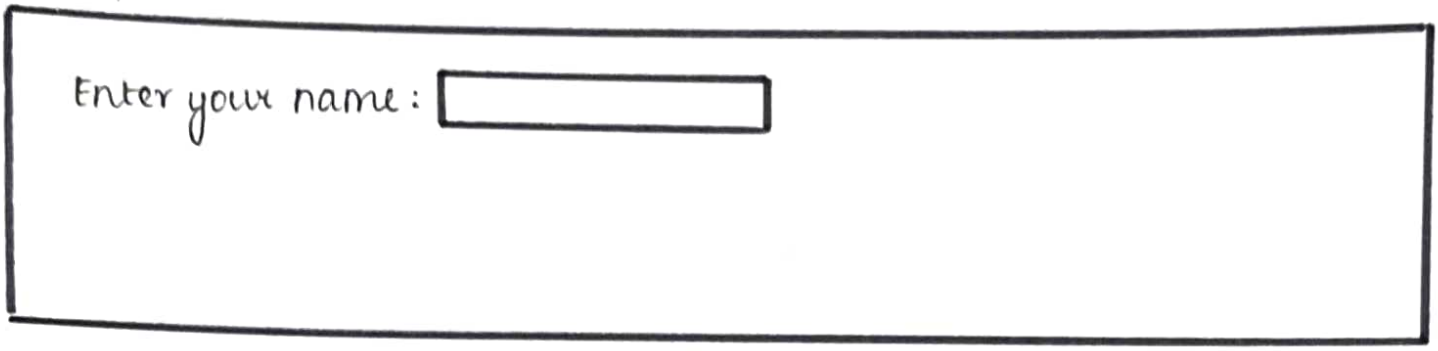
## React Forms

### Example

Add a form that allows user to enter name:

```
function Myform( ){
  return (
    <form>
      <label> enter name:
        <input type = "text"/>
      </label>
    </form>
  )
}
```

Output




## 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

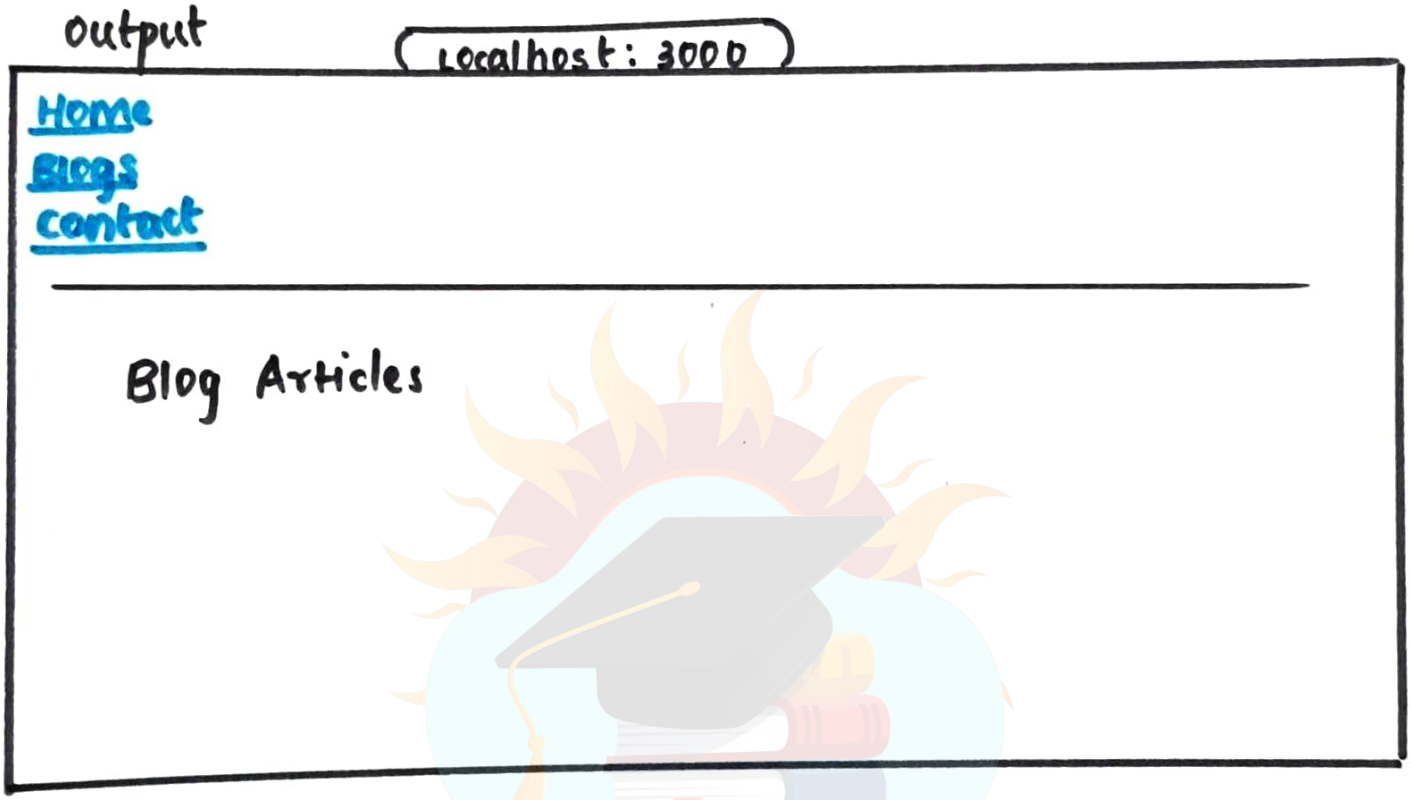
Now use router index.js file

### 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**




## 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

# React Memo

React Memo is Higher Order Component which itself wraps around component to rendered output

It is higher order component
uses last rendered result
only check for prop changes
Effect on React Hooks

## where to use React Memo

If you are using a pure functional component :
use it when you know beforehand that a component will render quite often.



use it if re-rendering is done using same props
use it if your component is big enough to have props equality check done by React with decent number of UI elements.

## where NOT to use React Memo?

You should not use React Memo in all React component that implements the shouldComponentUpdate ( ) method.
This is because it return true by default.

The render change that occurs by using ReactMemo is same implementation of shouldComponentUpdate( ) which does shallow comparison.

other than this

- dont use ReactMemo if component is'nt heavy
- wrapping a class based component in React Memo is undesirable

## 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>
   );
  }
}
```



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>
)
}
```

## Difference between state and props

| Props | state |
|---|---|
| The data is passed from one component to another | The Data is passed within the component |

| | |
|---|---|
| Props can be used with state and functional components. | The state can be used only with state component / class component |
| Props are read-only | The state is both read and write. |
| It is immutable | It is mutable. |
| props are controlled by whoever renders the component | state is controlled by react component |
| props can be accessed in functional component using props parameter and in-class component props can accessed using this.props | state can be accessed using the state hooks in functional components and in class components can be accessed |
| Props are read-only | state changes can be asynchronous |
| props communicate between components | state display changes with component |

# 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. |

# 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



Example

~~create style object named myStyle:~~

```
const Header = () => {
    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


## 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;

}
```

# Styling React Using Sass

## what is sass

Sass is a css pre-processor

Sass files are executed on server and sends css to browser

Install Sass by running this command on your terminal

```
>npm i sass
```

create a sass file with .scss

## Example

my-sass.scss

create a variable to define color of text

```
$myColor: red;

h1 {
  color: $myColor;
}
```

# React Hooks

## what are react hooks?

React Hooks are a new feature of react.js using this it is possible to use state and other react features. without writing class

## Rules to use hooks

Hooks should be used in topmost scope of code and never be used within loops, conditions or nested functions

Hooks should only be used by React function components



Dont use ordinary javascript methods to call hooks.

These rules are also applicable for custom hooks.

React has also built in hooks. ie useEffect and UseState.

## Use State Hook

The use state Hook is used for storing a state within a component.

The UseState hook allows you to store and access state inside a component without using this.state or this.setState().

## useEffect Hook

It gives function components the ability to perform side effects, resulting in accomplishing the same thing that componentDidMount, componentDidUpdate and ComponentWillUnmount do in React classes, but with single API

## Custom Hooks.

custom hooks is an effective option in case where we want to implement derived functionality of both usestate and useEffect

## Benefits of React Hooks.

Easy to understand complex components

Reduced complexity without classes.

Easy to reuse Stateful Logic.

## React Flux Vs MVC

MVC stands for Model view controller. It is an architectural pattern used for developing the user interface.



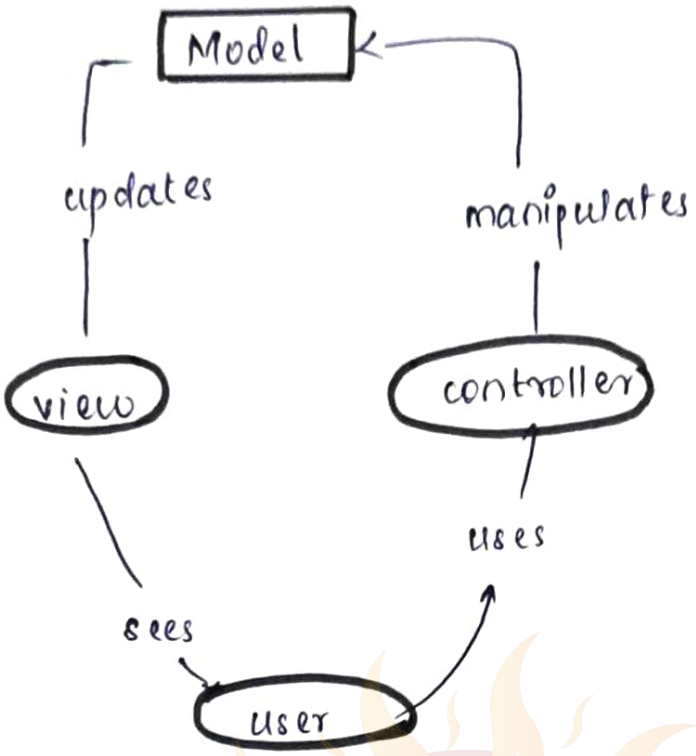
### MVC Architecture

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

View: It is used to display model in user interface

Controller: It acts an interface between model and view components.

It takes user input, manipulates the data and causes view




Mvc architecture

## 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

- Dispatcher
- Store
- Views

# Difference between MVC and Flux

© Topperworld

| MVC | FLUX |
| --- | --- |
| It is introduced in 1976 | It was introduced just few years ago |
| It supports bidirectional data flow model | It supports uni directional data flow model. |
| In this, data binding is key | In this, events are key |
| It is synchronous | It is asynchronous |
| It is hard to debug | It is easy to debug |
| It is difficult to understand as project size increases | It is easy to understand |
| Testing of application is difficult | Testing of application is easy |
| Scalability is complex | scalability is easy |

# 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

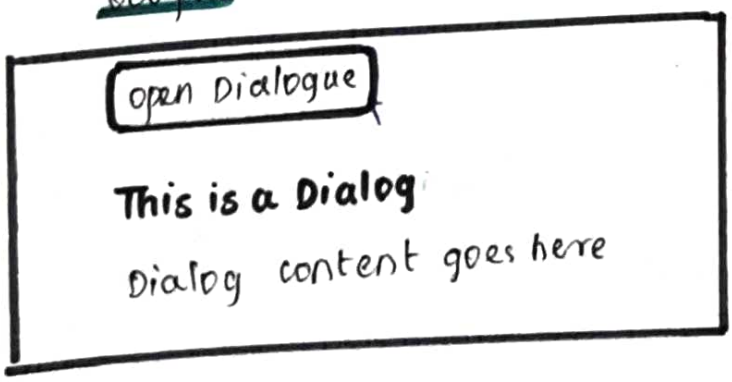
**Dialog**

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.

# 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 Java script 6

c. None

d. To render redux

20. why react is mainly used for

a. Database

b. MVC

c. Extention

d. **User interface**