

UNIT IV

React Basics

Contents

React: Introduction - Installation - create React app - components - state - props - props validation - state vs props - constructor - Component API - Component Life cycle - Forms - controlled and uncontrolled component - Events - conditional rendering

React: Introduction

- React JS is a JavaScript library for creating user interfaces
- React makes development of UI components easy and modular
- React JS was created by Jordan Walke, a software engineer at Facebook
- ReactJs is open source

Key features of Reactjs

Component-based: Components are the smallest unit in a React application. Anything that we want to render is rendered through components. Components help in maintainability

Virtual DOM: React uses virtual DOM concept for DOM manipulation which improves the performance of the application

Unidirectional data flow: React's one-way data flow (also called one-way binding) keeps everything modular and fast and easy for debugging.

JSX syntax: React used JSX syntax which is similar to XML and HTML syntax which makes it easy for writing markup and binding events in components

Applications

Who is using React?



Facebook



Prime video



Netflix



Instagram

Installation of Reactjs

1. Install the node version 14.0 or Higher
2. Give the following command in command prompt

`npx create-react-app projectname` (npm install -g npm) (npm install -g create-react-app)→npm installation

Example:

`npx create-react-app sample`

3. Navigate to the project folder in command prompt

`cd sample`

4. To run the react application give the command in command prompt as

`npm start`

5. Open browser and specify the url as

`localhost:3000`

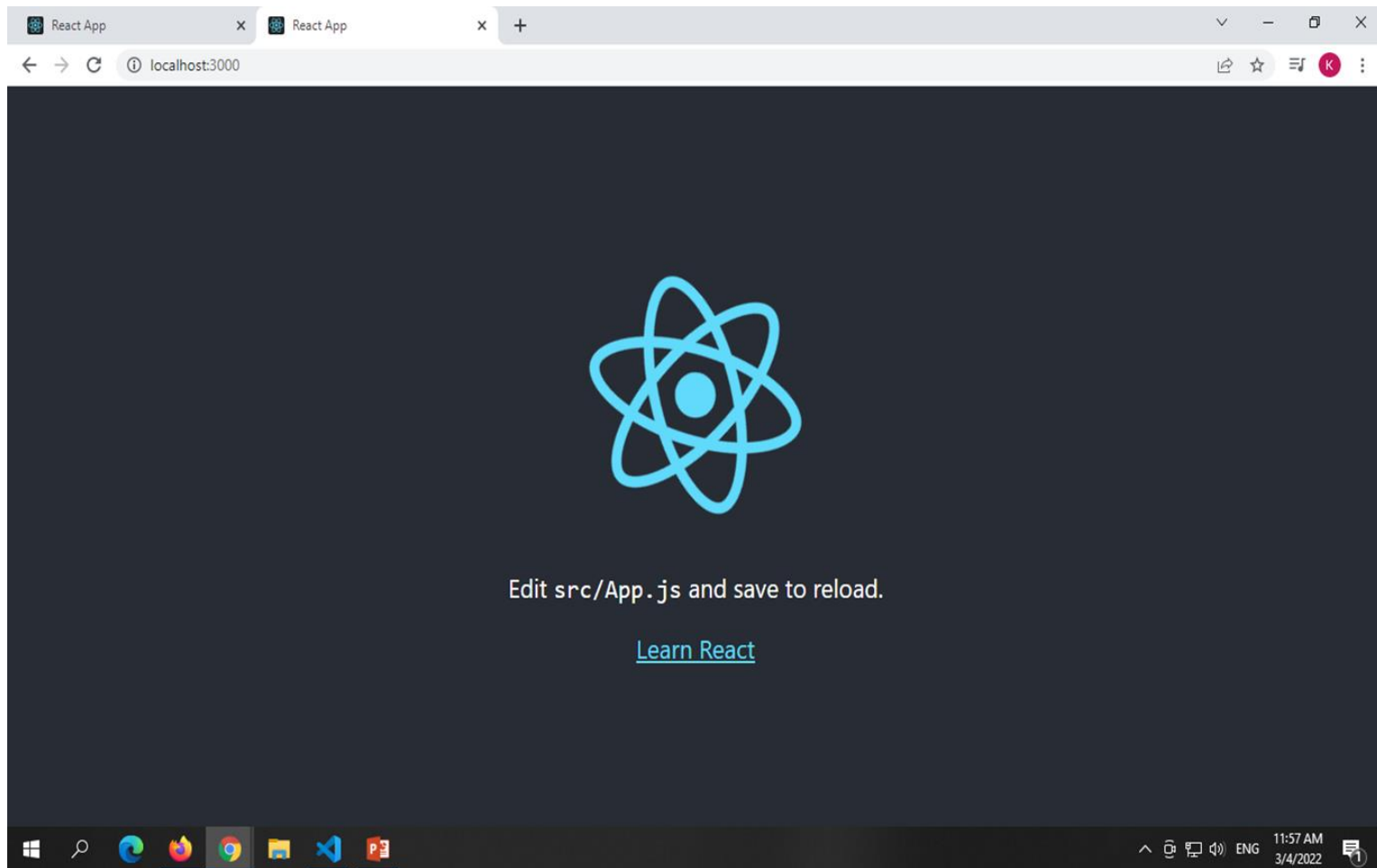
Note: React will run in the default port number 3000)

And if proxy denies installation try command

1. `npm config set registry http://registry.npmjs.org/`

2. `npm config rm proxy`

3. `npm config rm https-proxy`



Files in the react.js

1. Index.html is the first file gets run
2. App is the default parent component in react
3. Index.js says what are the components to be rendered
4. App.js says what content to be rendered in app component

Files	Purpose
node_modules	All the node module dependencies are created in this folder
public	This folder contains the public static assets of the application
public/index.html	This is the first page that gets loaded when we run the application
src	All application related files/folders are created in this folder
src/index.js	This is the entry point of the application
package.json	Contains the dependencies of the React application

JSX(JavaScript XML)

- JSX stands for JavaScript XML.
- JSX is syntax extension to javascript and is used to define the react components
- It allows us to define react elements using syntax that looks similar to HTML
- JSX can easily convert HTML tags to react elements.
- It is faster than regular JavaScript.
- JSX follows XML rule.

JSX Example 1:

```
function App() {  
  const name='welcome';  
  return (  
    <h1>{name}</h1>  
  
  );  
}  
export default App;■  
■
```

welcome

JSX Example 2:

```
import './App.css';  
function App() {  
  const name={  
    fname:'abi',  
    lname:'karthi'  
  }  
  return (  
    <div>  
      <h1>{name.fname}</h1>  
      <h1>{name.lname}</h1>  
    </div>  
  );  
}  
export default App;
```

abi

karthi

Note:

Inside the return statement if we want to return more than one element means put that in <div></div> tag because return will always return a single element

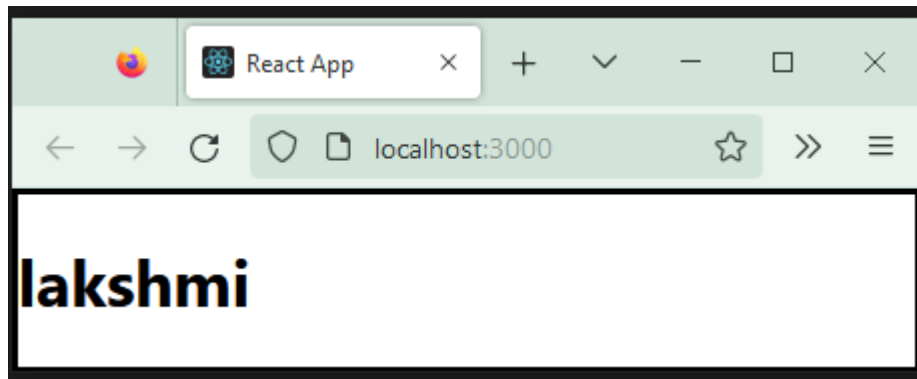
JSX Example 3:

```
import './App.css';  
function display()  
{  
  const name="lakshmi";  
  return name;  
}  
function App() {  
  return (  
    <div>  
      <h1>{display()}</h1>  
    </div>  
  );  
}  
export default App;
```

lakshmi

JSX Example 4:

```
function display()  
{  
  const name="lakshmi";  
  return name;  
}  
  
function App() {  
  return (  
    <div style={{border:'solid'}}>  
      <h1>{display()}</h1>  
    </div> );  
}  
  
export default App;
```



JSX Example 5:

```
import './App.css';  
function addition(a,b)  
{  
  var c=parseInt(a)+parseInt(b);  
  return c;  
}  
const a=window.prompt("enter a");  
const b=window.prompt("enter b");  
function App() {  
  return (  
    <div>  
      <h1>{addition(a,b)}</h1>  
    </div>  
  );}export default App;
```

localhost:3000

enter a

5

OK

Cancel

localhost:3000

enter a

10

OK

Cancel

Conditional statement JSX Example 5: localhost:3000

```
import './App.css';  
function max(a,b)  
{  
  if(parseInt(a)>parseInt(b))  
    return "a is greater";  
  else  
    return "b is greater";}  
const a=window.prompt("enter a");  
const b=window.prompt("enter b");  
function App() {  
  return (  
    <div>  
      <h1>{max(a,b)}</h1>  </div> );}  
export default App;
```

enter a

OK

Cancel

 localhost:3000

enter a

OK

Cancel

b is greater

Image in JSX:

```
import './App.css';  
function App() {  
  return (  
     );  
  }  
}
```

```
export default App;
```

Note:

Save your image file in **public** folder



Components

- React components are the fundamental unit of any React application.
- In this approach, the entire application is divided into a small logical group of code, which is known as components
- A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier
- Each component exists in the same space, but they work independently from one another and merge all in a parent component

Types of Components

- Functional Components
- Class Components

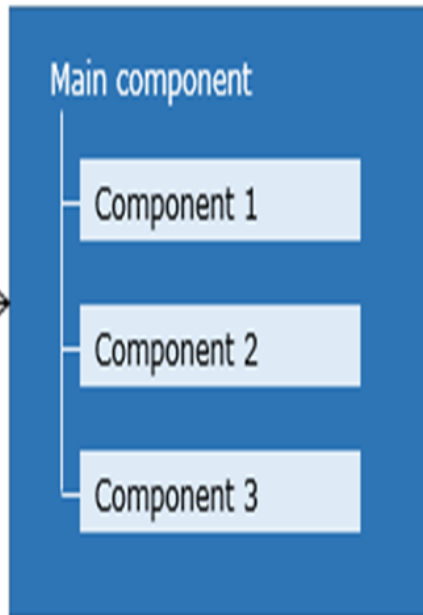
Components in Reactjs

React Components



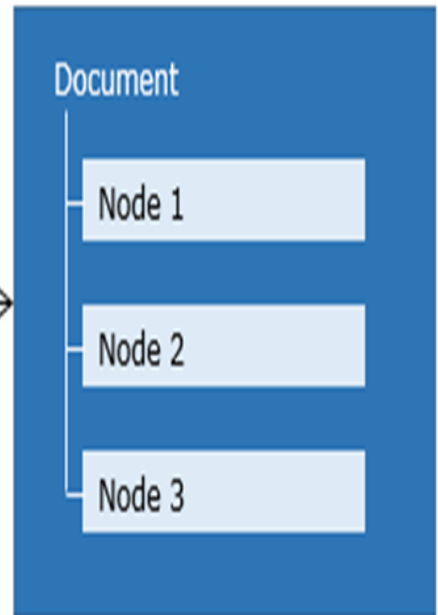
React components
we develop reside in
Virtual DOM

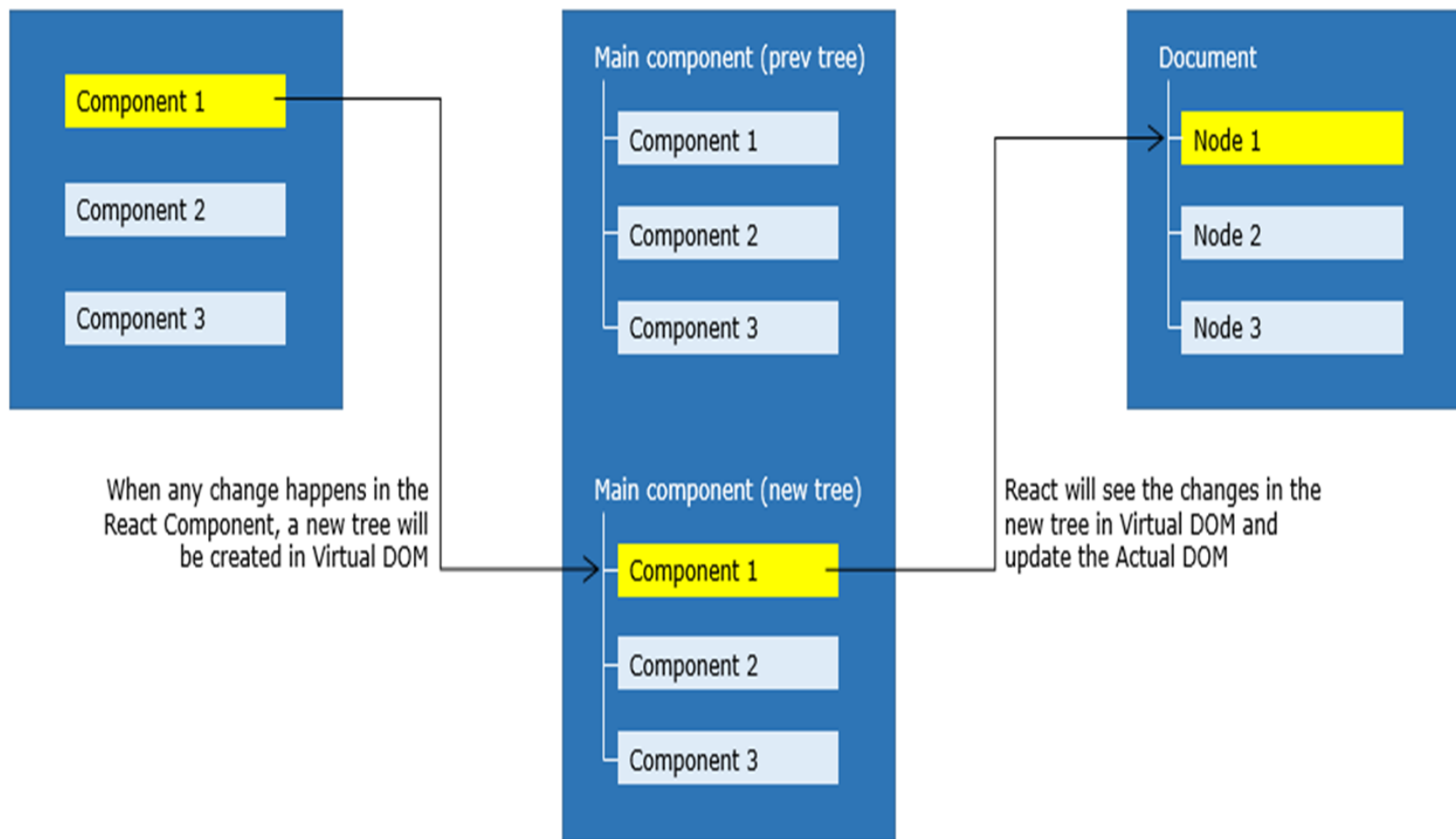
Virtual DOM



Virtual DOM
components will be
available as nodes in
Actual DOM

Actual DOM





- Virtual DOM is an abstraction of actual DOM where components are the nodes.
- We can programmatically modify virtual DOM by updating components.
- These updates are internally handled by React and in turn, updated in actual DOM.

1. Function Components

Function components are created with simple javascript function.

Step 1: Create a new js file with new component name (Example: User.js)

Step 2: Create a function inside User.js file with return statement.

Step 3: Export the newly created component

Step 4: Goto App.js file and include the import statement for newly created component (Example: import User from './User';)

Step 5: Inside App function include the newly created component

Note: All React component names must start with a capital letter

User.js

```
function User()  
{  
  return(  
    <h1> welcome</h1>  
  );  
  
}  
export default User;
```

Using arrow to define function

```
const User=()=>{  
  <h1> welcome to user page</h1>  
}  
  
export default User;■
```

App.js

```
import User from './User.js'
```

```
function App() {  
  return (  
    <div>  
        
      <User />  
    </div>  
  );  
}
```

```
export default App;
```



welcome

2. Class Components

A class component must include `extends React.Component`

The component requires `render()` method to return HTML

Step 1: Create a new js file with new component name (Example: Welcome.js)

Step 2: Create a class inside Welcome.js file with `render()`

Step 3: Export the newly created component

Step 4: Goto App.js file and include the import statement for newly created component (Example: `import Welcome from './Welcome';`)

Step 5: Inside App function include the newly created component

Welcome.js

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



App.js

```
import Welcome from './Welcome.js'  
function App() {  
  return (  
    <div>  
        
      <Welcome/>  
    </div>  
  );  
}  
export default App;
```



welcome

2. Nested Components

A component that is created and render inside another component is called as nested component.

```
import './App.css';  
function App() {  
  return (  
    <div classname="App">  
        
      <Nested/>  
    </div>  
  );  
}  
function Nested()  
{  
  return(  
    <h1>nested component</h1>  
  )  
}  
export default App;
```



nested component

Functional Components

- Simple functions
- Use functional components as much as possible
- Absence of 'this' keyword
- Solution without using state
- Mainly responsible for the UI
- stateless

Class Components

- More rich features
- Maintain their own private data-state
- Complex UI design
- Providing lifecycle hooks
- stateful

Props in react

- Props stand for "**Properties**." They are **read-only components**
- Props are arguments passed into React components
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes
- Props are **immutable** so we cannot modify the props from inside the component
- **props.children** is a special prop, automatically passed to every component, that can be used to render the content included between the opening and closing tags when invoking a component.

Props in functional component

User.js

```
const User= (props) =>{  
  return(  
    <div>  
      <h1>hai</h1>  
      <h1> welcome {props.name}</h1>  
    </div>)  
  }  
}
```

```
export default User; ■
```

■

App.js

```
import './App.css';  
import User from './User'  
function App() {  
  return (  
    <div classname="App">  
      <User name="dhivyaa" />  
    </div>  
  );}
```


```
export default App;■  
■  
■
```


hai

welcome dhivyaa

Props children in functional component

User.js

```
const User= (props) =>{  
  return(  
    <div>  
      <h1>hai</h1>  
      <h1> welcome {props.name}</h1>  
      <h1>{props.children}</h1>  
    </div>)  
  }  
  export default User; 
```



App.js

```
import User from './User'
function App() {
  return (
    <div>
      <User name="dhivyaa">
        <p> this is user 1</p>
        
      </User>
    </div>
  );
}
export default App;
```

hai

welcome dhivyaa

this is user 1



Props in Class component

User.js

```
import React from 'react'
class User extends React.Component{
  render()
  {
    return(
      <h1> hello {this.props.name}</h1>

    );
  }
}export default User;
```



App.js

```
import './App.css';
import User from './User'
function App() {
  return (
    <div classname="App">
      <User name="dhivyaa"/>

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

hello dhivyaa

State in reactjs

- React components has a built-in `state` object.
- The `state` object is where you store property values that belongs to the component.
- When the `state` object changes, the component re-renders.

Creating the state Object

- The state object is initialized in the constructor
- The state object can contain as many properties
- Refer to the state object anywhere in the component by using the `this.state.propertyname`
- Refer to the state object in the `render()` method
- To change a value in the state object, use the `this.setState()` method.

State in Class component

User.js

```
import React from 'react'
class User extends React.Component{
  constructor(){
    super()
    this.state={
      name: "dhivyaa",
      address:"erode"
    }
  }
  render()
  {
    return(
      <div>
        <h1> hello {this.state.name}</h1>
        <h1>address={this.state.address}</h1></div>
      );}}
export default User;
```

App.js

```
import './App.css';
import User from './User'
function App() {
  return (
    <div classname="App">
      <User/>
    </div>
  );}
```

```
export default App;■
```



hello dhivyaa

address=erode

setState in Class component

User.js

```
import React from 'react'
class User extends React.Component{
  constructor(){
    super()
    this.state={
      name: "dhivya",
      address:"erode"
    }
  }
  updateState=()=>
  {
    this.setState({
      name:"nithya",
      address:"chennai"
    })
  }
```

```
render()
{
  return(
    <div>
      <h1> hello {this.state.name}</h1>
      <h1>address={this.state.address}</h1>
      <button onClick={this.updateState}> click</button>
    </div>
  );
}
```


App.js

```
import './App.css';
import User from './User'
function App() {
  return (
    <div classname="App">
      <User />
    </div>
  );}
```

```
export default App;
```



hello dhivyaa

address=erode

click

hello nithya

address=chennai

click

Hooks in reactjs

- Hooks were added to React in version 16.8.
- Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.
- Hooks allow us to "hook" into React features such as state and lifecycle methods.
- The React useState Hook allows us to track state in a function component.
- State generally refers to data or properties that need to be tracking in an application.
- To use the useState Hook, we first need to import it into our component.

```
import { useState } from "react";
```

- useState accepts an initial state and returns two values:
 - The current state.

User.js

```
import React, { useState } from 'react';  
function User() {  
  const [count, setCount] = useState(0);  
  return (  
    <div>  
      <h1>You clicked {count} times</h1>  
      <button onClick={() => setCount(count + 1)}>  
        Click me  
      </button>  
    </div>  
  );  
}export default User;■  
■
```

App.js

```
import './App.css';  
import User from './User'
```

```
function App() {  
  return (  
    <div classname="App">  
      <User/>  
  
      </div>  
    );  
};
```

```
export default App;
```

You clicked 11 times

Click me

Constructor in Reactjs

- The constructor is a method used to initialize an object's state in a class
- The constructor in a React component is called before the component is mounted.
- **super(props)** method should be used before any other statement inside constructor
- If **super(props)** is not used then **this.props** will be undefined in the constructor
- In React, constructors are mainly used for two purposes:
 - It used for initializing the local state of the component by assigning an object to **this.state**.
 - It used for binding event handler methods that occur in your component.
- **Syntax:**

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

Props Vs State

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

Component Life Cycle

- In ReactJS, every component creation process involves various lifecycle method
- The lifecycle of the component is divided into **four phases**. They are:
 1. Initial Phase
 2. Mounting Phase
 3. Updating Phase
 4. Unmounting Phase

1. Initial Phase

In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

- **getDefaultProps()**

It is used to specify the default value of `this.props`. It is invoked before the creation of the component or any props from the parent is passed into it.

- **getInitialState()**

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

2. Mounting phase

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

- **componentWillMount()**

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

- **componentDidMount()**

This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.

- **render()**

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

3.Updating phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy.

componentWillRecieveProps()

- It is invoked when a component receives new props. This is invoked after this.setState() method.

componentWillUpdate()

- It is invoked just before the component updating occurs.

componentDidUpdate()

- It is invoked immediately after the component updating occurs

`shouldComponentUpdate()`

- It is invoked **when a component decides any changes/updation** to the DOM.

`render()`

- It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number.

4.Unmounting phase

- It is the **final phase of the react component lifecycle**
- It is called **when a component instance is destroyed and unmounted from the DOM**
- **componentWillUnmount()**
- This method is invoked immediately **before a component is destroyed** and unmounted permanently.

React Forms

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

Creating Form

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

There are mainly two types of form input in React.

1. Uncontrolled component
2. Controlled component

Uncontrolled component

- The uncontrolled input is similar to the traditional HTML form inputs.
- **Uncontrolled Components** are the components that are not controlled by the React state and are handled by the **DOM** (Document Object Model). So in order to access any value that has been entered we take the **help of refs**.
- The DOM itself handles the form data. Here, the **HTML elements maintain their own state** that will be updated when the input value changes.
- **To write an uncontrolled component**, you need to **use a ref to get form values** from the DOM.
- In other words, there is no need to write an event handler for every state update. You can use a ref to access the input field value of the form from the DOM.

User.js

```
import React, { Component } from 'react';
class User extends React.Component {
  constructor(props)
  {
    super(props);
    this.input = React.createRef();
    this.input1 = React.createRef();
  }
  updateSubmit=(event) =>{
    document.write("The name is "+this.input.current.value+"<br>");
    document.write("The company name is "+this.input1.current.value);
    event.preventDefault();
  }
}
```



```
render() {  
  return (  
    <form onSubmit={this.updateSubmit}>  
      <h1>Uncontrolled Form Example</h1>  
      <label>Name:  
        <input type="text" ref={this.input} />  
      </label>  
      <label>  
        CompanyName:  
        <input type="text" ref={this.input1} />  
      </label>  
      <input type="submit" value="Submit" />  
    </form>  
  ); } } export default User;
```

App.js

```
import User from './User'

function App() {
  return (
    <div classname="App">
      <User/>
    </div>
  );}
```

```
export default App;■
```



Uncontrolled Form Example

Name: CompanyName:

The name is John

The company name is TCS

Controlled component

- In the controlled component, the input form element is handled by the component rather than the DOM.
- Here, the mutable state is kept in the state property and will be updated only with **setState()** method.
- Controlled components have functions that govern the data passing into them on every **onChange event**
- When the **submit button** is clicked the data is saved to state and updated with **setState()** method.
- **preventDefault ()** is an event in react which is used to handle the default behavior in event handling mechanism.

It is also called as a "**dumb component**"

User.js

```
import React from 'react';  
class User extends React.Component {  
  constructor(props)  
  {  
    super(props);  
    this.state = {  
      username: '',  
      comments: '',  
      topic: ''    };  
  }  
  handleChange=(event)=> {  
    this.setState({username: event.target.value});  
  }  
}
```

```
handleCommentsChange=(event)=> {  
  this.setState({comments: event.target.value});  
}
```

```
handleTopicChange=(event)=> {  
  this.setState({topic: event.target.value});  
}
```

```
handleSubmit=(event) =>{  
  document.write("The name is "+this.state.username+"<br>");  
  document.write("The comment is "+this.state.comments+"<br>");  
  document.write("The Topic is "+this.state.topic+"<br>");  
  event.preventDefault();  
}
```

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <h1>Controlled Form Example</h1>  
      Name:<input type="text" value={this.state.username} onChange={this.handleChange} />  
      <br></br><br></br>  
      comments:<textarea value={this.state.comments}  
        onChange={this.handleCommentsChange}></textarea>  
      <br></br>  
      Select the Topics:<select value={this.state.topic} onChange={this.handleTopicChange}>  
        <option>react</option>  
        <option>vue</option>  
        <option>angular</option>  
      </select>  
      <input type="submit" value="Submit" />  
    </form>  
  ); } } export default User; █
```

App.js

```
import User from './User'

function App() {
  return (
    <div classname="App">
      <User/>
    </div>
  );}
```

```
export default App;█
```

```
█
█
█
█
█
█
```

Controlled Form Example

Name:

comments:

Select the Topics:

The name is C.R.Dhivyaa

The comment is Excellent Course

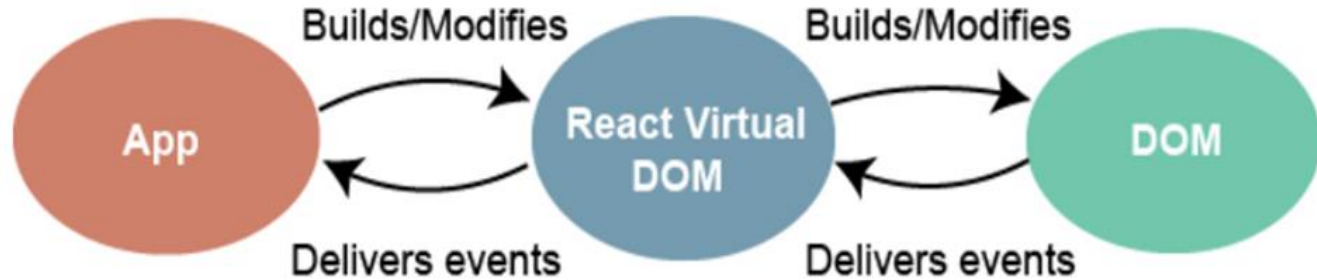
The Topic is vue

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

Events

- An event is an action that could be triggered as a result of the user action
- React has its own event handling system which is very similar to handling events on DOM elements.
- The react event handling system is known as Synthetic Events.
- SyntheticEvent allows events in React to easily adapt to different browsers.
- In react, we cannot return **false** to prevent the default behavior. We must call **preventDefault()** event explicitly to prevent the default behavior

Events Handler



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

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

User.js

```
import React from 'react';  
class User extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      collegeName: ''  
    };  
  }  
  changeText=(event) =>{  
    this.setState({  
      collegeName: event.target.value  
    });  
  }  
  handleSubmit=(event) =>{  
    document.write("The College name is "+this.state.collegeName+"<br>");  
  }  
}
```

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <h2>Simple Event Example</h2>  
      Enter your college name:  
      <input type="text" value={this.state.collegeName} onChange={this.changeText}/>  
      <input type="submit" value="Submit" />  
    </form>  
  );  
}  
}
```

export default User; ■

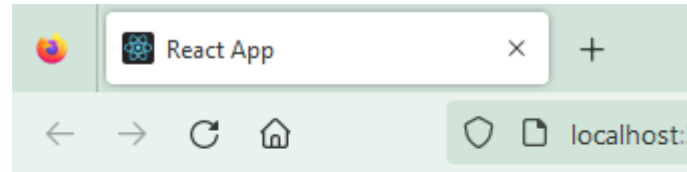
App.js

```
import User from './User'  
function App() {  
  return (  
    <div classname="App">  
      <User/>  
    </div>  
  );  
}
```

```
export default App;
```

Simple Event Example

Enter your college name:



The College name is KEC

Conditional rendering

- In React, conditional rendering works the same way as the conditions work in JavaScript.
- We use JavaScript operators to create elements representing the current state, and then React Component update the UI to match them.

Ways for conditional rendering in react

1.if-else

2.Ternary operator

3.Logical && operator

4.Switch Case Operator

1.if-else

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

Syntax:

```
if(Condition)
```

```
    statement;
```

```
else
```

```
    statement;
```

Example:

```
import './App.css';  
function max(a,b)  
{  
  if(parseInt(a)>parseInt(b))  
    return "a is greater";  
  else  
    return "b is greater";}  
const a=window.prompt("enter a");  
const b=window.prompt("enter b");  
function App() {  
  return (  
    <div>  
      <h1>{max(a,b)}</h1> </div> );}  
export default App;
```

localhost:3000

enter a

5

OK

Cancel

localhost:3000

enter a

10

OK

Cancel

b is greater

2. Ternary operator

Syntax:

condition ? true : false

Example:

```
import './App.css';  
function max(a,b)  
{  
  return (a>b)? "a is greater":"b is greater";  
}  
const a=window.prompt("enter a");  
const b=window.prompt("enter b");  
function App() {  
  return (  
    <div>  
      <h1>{max(a,b)}</h1> </div> );  
}
```

localhost:3000

enter a

5

OK

Cancel

localhost:3000

enter a

10

OK

Cancel

b is greater

3.Logical && operator

Syntax:(expression 1 && expression 2)

function max(a,b,c)

```
{  
  if (a>b && a>c)  
    return "a is greater";  
  else if(b>c)  
    return "b is greater";  
  else  
    return "c is greater";  
}
```

```
const a=window.prompt("enter a");
```

```
const b=window.prompt("enter b");
```

```
const c=window.prompt("enter c");
```

```
function App() {  
  return (  
    <h1>{max(a,b,c)}</h1>  
  );  
}  
export default App;■  
■
```

localhost:3000

enter a

50

OK

Cancel

localhost:3000

enter c

70

☐ Prevent this page from creating additional dialogs

OK

Cancel

localhost:3000

enter b

100

☐ Prevent this page from creating additional dialogs

OK

Cancel

b is greater

4. Switch Case Operator

```
function number(b)
{
  switch(b)
  {
    case 1: return("The entered number is 1");
    case 2: return(" The entered number is 2");
    default: return("The entered number is other than 1 and 2")
  }
}

var a=window.prompt("enter the number");
var b=parseInt(a);
function App() {
  return (
    <h1>{number(b)}</h1>
  );
}
export default App;
```



A screenshot of a web browser window with the address bar showing 'localhost:3000'. Below the address bar, there is a text input field with the placeholder text 'enter the number'. The input field contains the number '1'. To the right of the input field, there are two buttons: 'OK' (blue) and 'Cancel' (gray).

The entered number is 1