Andhra Pradesh State Skill Development Corporation

Web designing using ReactJs

ReactJS

What is ReactJs?

ReactJs is once an open-source JavaScript library for building user interfaces specifically for single-page applications. React allows us to create reusable UI components. It is developed & maintained by Facebook. React can be used as a base in the development of single-page application

What is the Reason Behind JavaScript Developers Using React JS?

While building an application, User-interfaces are the collection of on-screen menus, search bars, buttons & anything else a user interacts with the web application. Before React JS, developers used to write raw JavaScript language on its own or by using less UI-focused React predecessors like JQuery, which takes much more development time and is quite risky for the opportunities of bugs and errors.
So, in 2011, Facebook engineer Jordan Walke created React JS, to improve the UI development

Is React JS Front-end or Back-end? Why You Should Use React JS?

React is used as client-side programming building things that a user will see and interact on-screen in their browser window which makes React JS a front-end library. Here are some of the reasons below for using React JS:

1. The React JS development techniques are much simpler to grasp because of its component-based approach. ReactJS characteristics to learn and build a professional web (and mobile applications).
2. It is easy to learn with just previous basic programming knowledge.
3. The applications made by React JS are easy to test and can be treated as functions, manipulate the React JS view and take a look at the output, triggered actions, functions, events, etc.

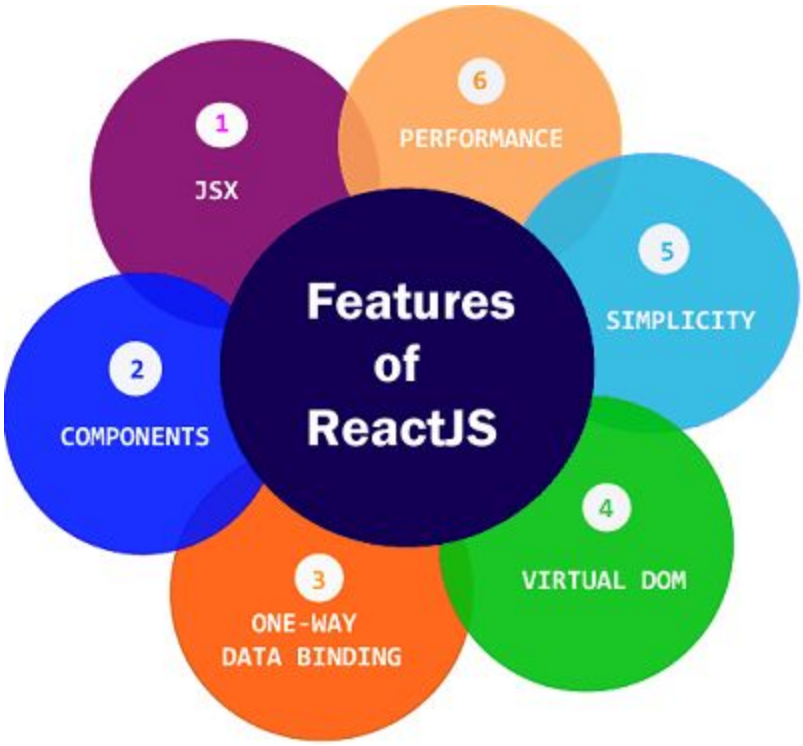
What Are React JS Key Features?

JSX(JavaScript XML): JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React. It is the short-form of JavaScript extension (a React extension) acting as a simple JavaScript that allows HTML quoting and uses these HTML tag syntax to render subcomponents. JSX helps developers to modify the Document Object Model (DOM) of the website where DOM is a representative tree of how the website is arranged

Virtual DOM: If a developer uses JSX to manipulate and update its DOM, React JS creates an in-memory data structure cache known as “Virtual Document Object Model”. VDOM is a copy of the actual/real DOM of the website and React JS uses the virtual DOM to change the data and then updates in the browser. React JS scans the Virtual DOM to see what the actual changes made by the user and selectively updates that section of the DOM only. This process ensures less loading time & computing power

React Native: React has native libraries that were generally announced by Facebook in 2015, which provides react architecture to native applications like iOS & Android. Hire React Native Developers today & build an amazing mobile application for your business

Single-Way Data Flow: In React, a set of immutable values is passed to the components renderer as properties in its HTML tags. The component cannot directly modify any properties but can pass a callback function with the help of which we can do modifications. This complete process is known as “properties flow down; actions flow up”.



Prerequisites

- Node (nodejs.org)
- Knowledge on
  - HTML5
  - CSS3
  - JavaScript (ES6)

Babel && WebPack

- All the browsers support ES-5.
- We are using ES-6 concepts
- Babel is a transpiler that converts the new JS code to Older ones.
- Minify (CSS, SCSS, SASS, JavaScript)

- WebPack consists of two functionalities
    - Loaders
        - CSS-Loader
        - SASS-Loader
        - Babel-Loader
    - Plugins

Installation

```
node -v
npm -v
npm install create-react-app -g
create-react-app <project_name>
npx create-react-app <project_name>
```

Components in React

What is Component?

Components are independent and reusable bits/snippets of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via render() function. Components come in two types, Those are

1. Class components
2. Function components

Class components

- For representing a class component we have to use class keyword
- render() for returning a statement
- We have to acquire the properties from the base class for implementing.

Functional components

- We have to use function | ()=> for implementing a functional component
- Can return a statement directly
- High performance than class Component

Styling components

```
// Inline styles
 <h2 style={{backgroundColor:"green"}}> Sample content </h2>`
```

```
// Styling component using JavaScript object
 var style={
   backgroundColor: "reg",
   color: "#000"
 }
```

Applying styles dynamically

```
import React,{useState} from 'react';
import './App.css';

function App(){
 var style={
   background:"green"
 }

 var changeBackground=(e)=>{
   e.target.setAttribute('class','changeBackground')
 }

 const [data,setData]=useState({"name":"Hanuman","role":"Full stack developer"})
 return (
   <>
     <h2 onMouseOver={(e)=>{changeBackground(e)}}> {data.name} is working as
{data.role}.</h2>
   </>
 )
}

export default App;
```

Composition

The process of combining multiple components together

React fragment

We have to use react fragments to avoid a number of divisions.

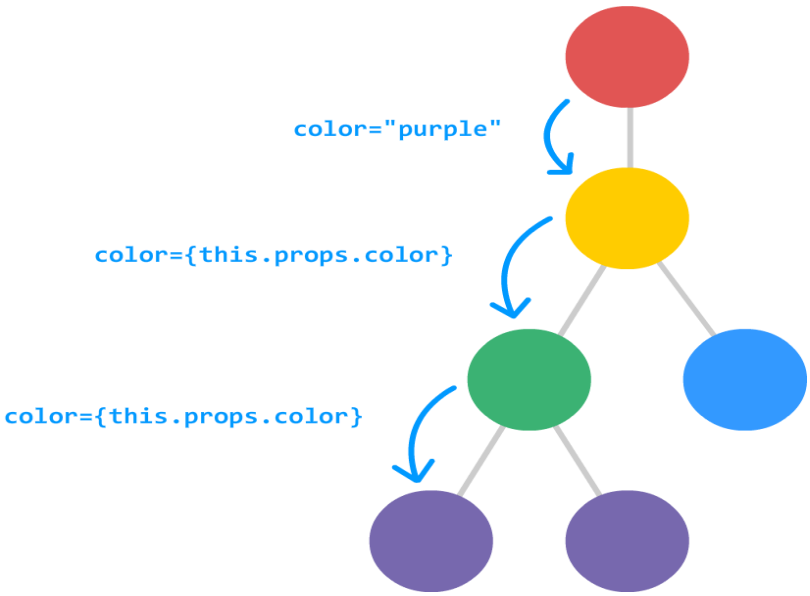
```
<React.Fragment>
   <Header />
   <h2> Sample heading </h2>
</React.Fragment>

  or
```

```
<>
  <Header />
  <h2> Hellow world </h2> <h2> Hi everyone</h2>
  <App />
 </>
```

Props and State in React

Props in React

- **P**rops is a special keyword in React which is used for passing the data from one component to another component.
- It is unidirectional
- props data is read-only, which means the data coming from the parent component should not be changed by child components.
- Props are arguments passed among React components.



Example: (Passing properties (props) from class component to a functional component) App.js

```
import React,{Component} from 'react';
import './App.css';
import StatesInFunction from './StatesInFunction';

class App extends Component{
```

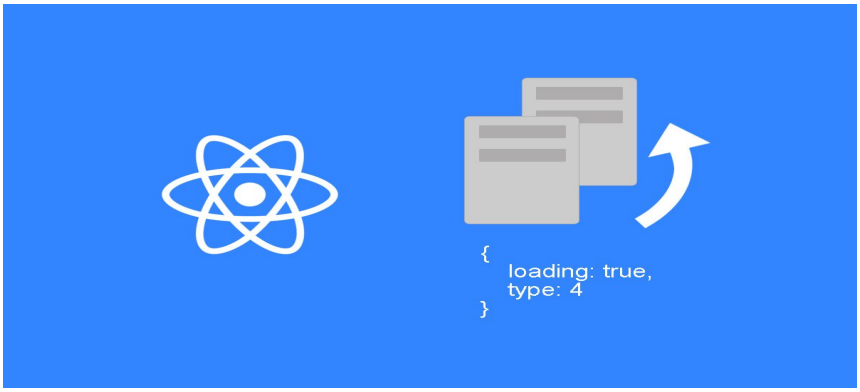
```
        render(){
                return (
                        <div className="App">
                                // Props
                                <StatesInFunction name="Nitesh" age="20 years"/>

                        </div>
                        )
                }
        }

export default App;
```

But how we are going to access the props in class components

Here we go with that

```
import React from 'react';

        export default class StatesInFunction extends React.Component{
                render(){
                return(
                        <div>
                                <h2> {this.props.name} is {this.props.age} old. </h2>
                        </div>
                )
                }
        }
```

State provides an ability to store information in the components. Within the class component only we can use the state concept. We've to use the constructor method as well as the super method for initializing the state as we need information from the base class. By using this.state we can initialize a state. The state should be in object format.



Syntax:

```
constructor(){
        super();
        this.state={
        name:"Hanuman"
        }
}
```

We can implement the state concept in class components only.

Manipulating data from a state

```
changeStateValues=()=>{
   this.setState({
     "name":"Rajesh",
     "role":"MERN developer"
   })
 }
 render(){
   return(
   <>
     <Header />
   <h2 onMouseOver={this.changeStateValues}> {this.state.name} working as a {this.state.role}
</h2>
     <App />
   </>
   )
 }
```

Using prevState for manipulating the data of a state

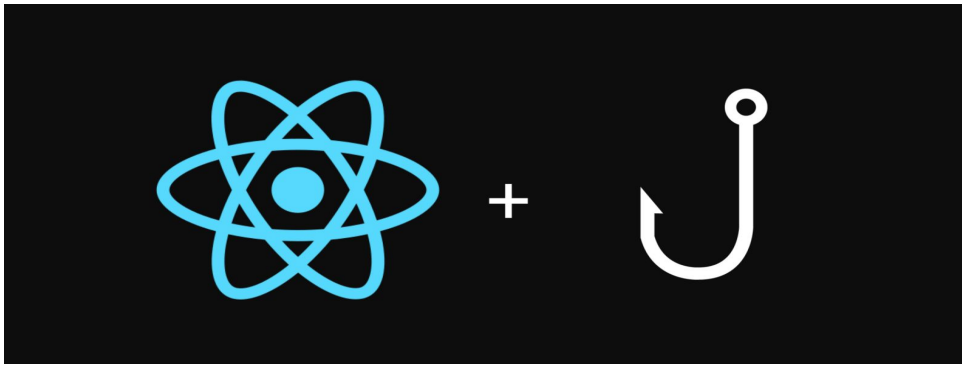
```
incrementCounter=()=>{
   this.setState(prevState=>(
     {counter: prevState.counter+1}
   ))
 }
```

Decrementing the value of the state

```
decrementCounter=()=>{
   if(this.state.counter>0){
   this.setState(prevState=>(
     {counter: prevState.counter-1}
   ))
   }
 }
```

Hooks:

Hooks provides a way to use the features of class components in the functional components



useState: By using this hook we can implement states functionality in functional components. For declaring this hook, we've to use a destructuring method.

Example:

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

   function HookExample(){
   let text=`Hi everyone! How are you?`;
   let maxLength=10;

   const [hide,setHidden]=useState(true);
```

```
  return(
     <>
        {hide ? `${text.substr(0,maxLength).trim()}...`: text} <br></br>
        {hide ? (<button onClick={()=>setHidden(false)}> Read entire text </button>) :
        (<button onClick=      {()=>setHidden(true)}> Read less text </button>)}
     </>
   )
 }

export default HookExample;
```

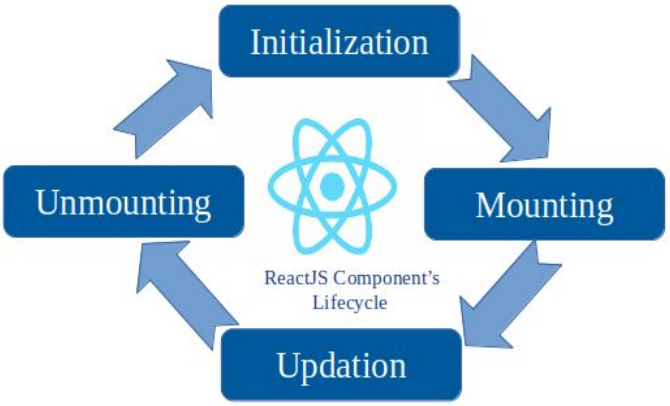
Using hooks concept for implementing state functionality in functional component

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

function App(){

 var initialData=()=>{
  setData({
   name:"Hanuman",
   role:"Full stack developer"
  })
 }

 const [data,setData]=useState({"name":"Hanuman","role":"Full stack developer"})
 return(
  <>
   <h2 onMouseOver={()=>{setData({name:"Rajesh",role:"MERN developer"})}}
onMouseOut={initialData}> {data.name} is working as a {data.role} </h2>
  </>
 )
}

export default App;
```

Life cycle methods of React

Four phases of a React component

- Initialization
- Mounting
- Updating
- Unmounting



Initialization It is the birth phase of the lifecycle of reactJs component. From here , the component starts its journey on a way to the DOM.

- Default Props
- Initialization State

Setting default props in component

```
import React from 'react';

  class Example extends React.Component{
  constructor(props){
     super();
     this.state={
        name: " "
     }


  }

  render(){
     return(
        <div>
           <h1> {this.state.name} </h1>
           <h2> {this.props.name}</h2>
        </div>
     )
   }
  }

  Example.defaultProps={
     name: "Kalyan",
```

```
    lastname:"Chakravarthi"
  }

  export default Example;
```

Setting default state in component.

```
constructor(props){
      super();
      this.state={
         name: " "
      }
     }
```

Mounting

- componentDidMount()

  This function gets invoked once after the render function is executed for the first time.

- componentWillMount()

  This function gets invoked once before the render method is executed.

```
import React from 'react';

  export default class Practice extends React.Component{
     constructor(){
        super();
        this.state={
        name:"Hanuman"
        }
     }

     componentWillMount(){
        this.setState({
           name:"Ninad"
        });
        console.log(this.state.name);
     }

     componentDidMount(){
        this.setState({
           name:"Hi"
        })
        console.log(this.state.name);
```

```
    }

    render(){
        return(
            <div>
                <h1> {this.state.name} </h1>
            </div>
        )
    }
}
```

Updating

- **render()**

  As I have talked about earlier, render() is the most used method for any React powered component which returns a JSX with backend data. It is seen as a normal function but the render() function has to return something whether it is null. When the component file is called it calls the render() method by default because that component needs to display the HTML markup or we can say JSX syntax.

- This method is required for returning a value in the class component.
- This is method is mandatory

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

        class App extends Component {
                render() {
                        return (
                                <div>
                                <h1 className="App-title">Welcome to React</h1>
                                </div>
                        );
                }
        }

        export default App;
```

Please take a note here; we must return something, if there is no JSX for the return then null would be perfect, but must return something. In that scenario, you can do something like this.

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

```
class App extends Component {
        render() {
                return null;
        }
}

export default App;
```

- we cannot define `setState()` inside `render()` function. **Why???** Because `setState()` function changes the state of the application and causing a change in the state called the `render()` function again.
- So if you write something like this then calling the function stack will go for infinity and the application gets the crash.
- You can define some variables, perform some operation inside render() function, but never use the setState function. In general cases, We are logging out some variable's output in the render() method. It is the function that calls in mounting lifecycle methods.

- **shouldComponentUpdate()**
  - It will returns a boolean value (true or false)
  - It returns true value by default
  - The boolean value that specifies whether React will continue with rendering or not.
  - You can tell React not to render rows that do not need to be using this method.

```
class ListItem extends Component {
        shouldComponentUpdate(nextProps, nextState) {
        return nextProps.isFavourite != this.props.isFavourite;
        }
}
```

- **componentDidUpdate()**
  - This method is called when the information is updated in DOM.

```
import React from 'react';

export default class Work extends React.Component{
    constructor(){
        super();
        this.state={
            favoriteFood:"Potato fry"
        }
    }
```

```
    changeFood=()=>{
        this.setState({favoriteFood:"Carrot fry"})
    }

    componentDidUpdate(){
        document.getElementById("newInfo").innerHTML="My favorite food is
"+this.state.favoriteFood;
    }

    render(){
        return (
            <>
             <h2> {this.state.favoriteFood} </h2>
             <h3 id="newInfo"> </h3>
             <button onClick={this.changeFood}> Change Food </button>
            </>
        )
     }
}
```

- **getDerivedStateFromProps()**
    - This is the first method that is called when a component gets updated
    - We can set the state value by using initial props

**Work.js*

```
import React from 'react';

export default class Work extends React.Component{
   constructor(){
      super();
      this.state={
         favoriteFood:""
      }
   }

   static getDerivedStateFromProps(props,state){
      return {favoriteFood:props.food}
   }

   render(){
      return (
         <>
   <h2> My favorite food is {this.state.favoriteFood} </h2>
         </>
```

```
      )
    }
}
```

App.js

```
import React,{Component} from 'react';
import Work from './components/Work';

class App extends Component{
  render(){

    return (
      <div className="App card">

        <Work food="Chicken"/>
      </div>
    )
  }
}


export default App;
```

- **getSnapshotBeforeUpdate()**
  - By using this method we can access the state and props values before updating.

```
import React from 'react';

export default class Work extends React.Component{
    constructor(){
        super();
        this.state={
            favoriteFood:"Potato fry"
        }
    }

    componentDidMount(){
        setTimeout(()=>{
            this.setState({favoriteFood:"Chicken fry"})
        },3000)
    }

    getSnapshotBeforeUpdate(prevProp,prevState){
```

```
      document.getElementById("prevValue").innerHTML="The previous state value is
"+prevState.favoriteFood;
   }

   render(){
      return (
         <>
   <h2> My favorite food is {this.state.favoriteFood} </h2>
   <h3 id="prevValue"> </h3>
         </>
      )
   }
}
```

Pure Components in React:

What is a Pure component?

Pure Components in React are the components which do not re-renders when the value of state and props has been updated with the same values. If the value of the previous state or props and the new state or props is the same, the component is not re-rendered. Pure Components restricts the re-rendering ensuring the higher performance of the Component

(or)

Pure Component is one of the most significant ways to optimize React applications. The usage of Pure Component gives a considerable increase in performance because it reduces the number of render operation in the application

Features of React Pure Components:

- Prevents re-rendering of Component if props or state is the same
- Takes care of Updating the component implicitly
- State and Props are Shallow Compared
- Pure Components are more performant in certain cases

Higher Order Components (HOC's):

What are Higher Order Components(HOC's)?

A higher-order component is a function that takes a component and returns a new component.

Example:

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

```
var newData = {
  data: 'Data from HOC...',
}

var HOC = ComposedComponent => class extends React.Component {

    this.setState({
      data: newData.data
    });
  render() {
    return <ComposedComponent {...this.state} />;
  }
};
class MyComponent extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.data}</h1>
      </div>
    )
  }
}
export default HOC(MyComponent);
```

Routing in React (react-router-dom)

For implementing Routing in react, we have to install a package named react-router-dom. We have to follow the following command to install it.

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

Primary components of React-router-dom:

- BrowserRouter

  Is useful for initializing the navigation context. This is the parent component of react-router-dom

```
<BrowserRouter>
</BrowserRoputer>
```

- **Route**

  Is for specifying the path.

```
<Route exact path="/about" component={About}> </Route>
```

```
<BrowserRouter>
        <Route exact path="/about" component={About} />
</BrowserRouter>
```

- **Link**

  This is for providing events that calls to the path specified in the path of the route component.

```
        <Link to="/about"> Click me </Link>
        // <a href="/about"> Click me </a>
```

Example:

```
    <BrowserRouter>
        <Route path="/about" component={About}/>
    <BrowserRouter>

    <Link to="/about" />
```

Most of the web developers or web designers use the link attribute (<a>) to pass the data from one page to another page.

In react we are using the <Link> </Link> attribute instead of anchor tag (<a>).

But how we are going to pass data from one component to another component using <Link /> ?
For this we have to pass the parameters in the format of object as shown below.

```
<Link to={{pathname:"/resume", data:{id:index}}} className="button"> View profile </Link>
```

- Here the pathname specifies the url we are going to navigate.
- data represents the information which we are passing. Here id is the key and the index is the value.

If you want to access that information in destination component, we've to use the location keyword.

```
import React from 'react';
        import {profiles} from './data.json';
        import './App.css';

        let Resume=(props)=>{
        var info=profiles[props.location.data.id];
        return(
        <section className="parent">
                <article className="basicsCard">
                        <h2> {info.basics.name} </h2>
                </article>
        </section>
                )
        }

        export default Resume;
```

Regular expression in React: (*Program*)

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

export default function FormInput(){
   function testMobile(e){
      var reg=/^\S+@\S+[.]\S{1,}$/
      if(reg.test(e.target.value)){
         setContent({
            background:"green",
            text:"Valid Email id"
         })
      } else {
         setContent({
            background:"red",
            text:"Invalid Email id"
         })
      }
   }
   var [content, setContent]=useState({background:"#fff",text:"hi"})
   return(
      <div>
         <article style={{background:content.background, textAlign:"center"}}> {content.text}
</article> <br />
         <form>
            <input type="text" placeholder="Email id" onKeyUp={(event)=>{testMobile(event)}}/>
```

```
        </form>
      </div>
    )
}
```

Live search from JSON

The below example guides you through the concept of JSON, rendering JSON in react and the functionality of live-search using JSON data.

```
import React from 'react';
import './App.css'
import {profiles} from './data.json'
export default class Search extends React.Component{
   constructor(){
      super();
      this.state={
         search:""
      }
   }

   searchData=(e)=>{
      this.setState({search: e.target.value})
   }
   render(){
      const {search} =this.state;
      console.log(search)
      const filteredNames=profiles.filter(profileInfo=>{
         return profileInfo.name.toLowerCase().indexOf(search.toLowerCase())!==-1
      })
      return(
         <div className="container">
            <input type="text" placeholder="Search..."
onKeyUp={(event)=>{this.searchData(event)}}/>
            <div className="parent">
               {
                  filteredNames.map((item,index)=>(
                     <div className="child" key={index}>
                        {item.name}
                     </div>
                  ))
               }
            </div>
         </div>
      )
   }
}
```