# React JS

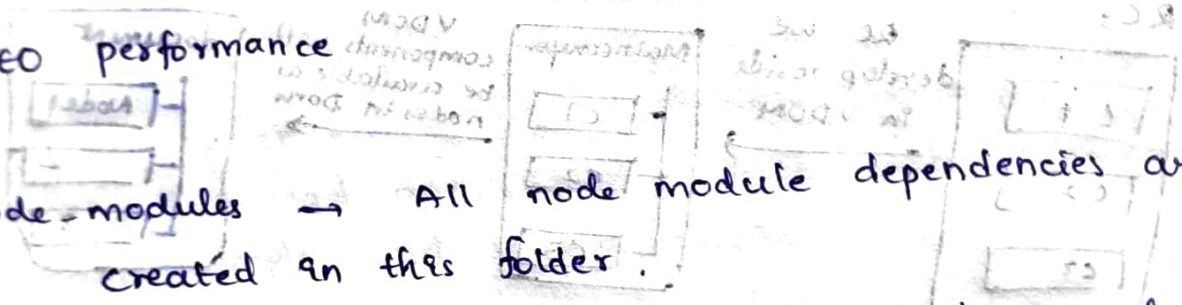
React JS library that optimizes DOM manipulation by writing very simple code.

JS library for creating UI, makes development of UI components easy and modular

→ used for applications where data keeps changing very frequently

## Key features

- Component-based architecture
  - anything you want to render on browser can be rendered through components.
- Virtual DOM - improves performance
- Unidirectional data flow (one-way binding) keeps everything modular fast and easy
- JSX syntax - used JSX ≈ XML, HTML
  - makes it easy for writing markup and binding events in components
- SEO performance



node_modules → All node module dependencies are created in this folder.

public/ index.html → first page that gets loaded when you run your application

src/ index.js → entry point of the application

package.json → contains dependencies of react application

# React components (RC)

**Why?**

When you have different sections in your page

challenges:

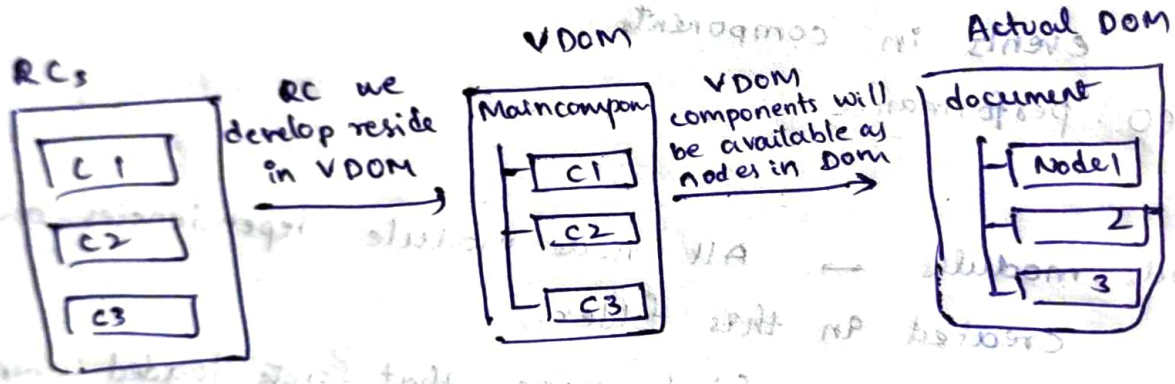
→ Entire page will get re-rendered even when a section of the page undergoes changes

→ You will have to rewrite code even they have similar behavior

→ You have to make sure that functionality of one part of the application do not interfere with another part.

**sol:** divide them into components encapsulate its own data and presentation logic then you can achieve modularity and re-usability.

**RC** fundamental unit of any React application
↳ works in conjunction with each other




VDOM → abstraction of VDOM

components = nodes

can be updated → modify VDOM

modifying is expensive

components

- class based
- Functional based
  - → simple JS func
  - → returns HTML ele

naming convention - Pascal

To display ele of a component, component must be rendered. for rendering a component,

ReactDOM.render( <param1 />, param2 )

param1 → name of component to be rendered

param2 → HTML node reference where component to be rendered

ReactCreateElement("h1", { }, "content")

"h1" → element

{ } → attrn

export → component must be exported so that it could be used in any other files.

components return JSX elements

## JSX (JavaScript XML)

Code using JS - ✓ write more lines of JS code
✓ difficult to understand
✓ productivity goes down

JSX in React - to create elements that are very easy to read and write, which makes component's code simple and understandable.

To create a login component using JSX:

```
function App() {
    return( <form> <h2> Login </h2>
        <input type="text" placeholder="Name" /> <br/>
        <input type="password" placeholder="pwd" /> <br/>
        <input type="submit" value="log" />
    </form>);
}
export default App;
```

JSX - allows you to write HTML like code with in JS code. It makes it easier to describe the structure of UI components.

React components often return JSX elements.

JSX code

```
<h1> Hello !! </h1>
```

conversion →

JS code

```
React.createElement("h1", {},
                    "Hello!");
```

This conversion happens, as browser doesnot understand JSX code, using plugins of the babel.

JS compiler transforms modern JS code into older version.

While rendering, you must render only one element. Adjacent JSX elements must be wrapped in an enclosing tag.  tag can be used for this.

## React Fragments

using this introduces an extra and unnecessary node into the DOM.

Sol: React Fragments allows to group a list of React elements without adding any extra node to the DOM.

```
function App() {

    return (
        <React.Fragment>
        <h3> ReactJS: </h3>
        <img src=" ... " width="120" hight="120" />
        <p> This is a paragraph </p>
    </React.Fragment>
    );
}
export default App;
```

Instead of React.Fragment → you can use <> empty tags

JavaScript Expressions in JSX

```
<h1> { Expression to be evaluated } </h1>
<h1> {" content to be displayed "} </h1>
```

Keys in React are used to identify the items that are added, removed or modified. Keys should be provided to the elements inside the array to provide a unique identity.

Keys should be unique.

map() → You can renders lists in React using map()

```
var employees = [
    { eId : 123 , name : "A" , desig : "SE" },
    { eId : 124 , name : "B" , desig : "SSE" },
    { eId : 125 , name : "C" , desig : "TA" },
];
```

→

```
return(
    <>
      <table>
        <thead><tr>
            <th> EmpId </th> <th> name </th>
            <th> Designation </th>
          </tr>
        </thead>
        <tbody>
          {employees.map((employee) => {
            return (
              <tr key={employee.empId}>
                <td> {employee.empId} </td>
                <td>          .name </td>
                <td>          .designation </td>
              </tr>
            );
          })}
        </tbody>
      </table>
    </>
  );
}
```

## Styling Components

✓ Inline styles

```
<h1 style={{ color: 'green' }}> Welcome </h1>
```

✓ Modifying App.css file

```
<h1 className="heading1"> Heading </h1>
```

In App.css

```
.heading1 {
    background-color: blue;
    color: white;
    border-radius: 10 px;
}
```

import App.css into App component

✓ can style components using bootstrap library,

install bootstrap library

```
npm install bootstrap

> node_modules
    > bootstrap
        > dist
            > css
                # bootstrap.min.css   ← import this
```

✓ React-Bootstrap library → library of reusable front end components

```
npm install react-bootstrap
```

✓ Material-UI - popular React UI framework which provides various components and themes for styling React components.

✓ index.css acts as a global css file

## State and props

In a real-time application, components must deal with dynamic data.

The data could be something internal to the component or may be the data is passed from another component. To bind the data to the component — state and props.

**State** - JS object used to manage data of an application
- initial value set for a component, which is used for interactivity.

```
constructor() {
    super();
    this.state = { counter : 1 };
}
```

Creating a timer component, where on clicking a button, the timer starts
button is invoked clicked, invoke handleClick() method - set the interval and pass it to start() method

```
handleClick(e) {
    this.interval = setInterval(this.start, 1000);
}

start() {
    this.setState( { secondsElapsed : this.state.secondsElapsed+1
                                                    } );
}
```

if it is called it calls render() method to render updated value on UI.

**setState()** is asynchronous
use callbacks to resolve issues.

**useState()** → to associate state with components

```
const [count, setCount] = useState(0);
```

count ← state var
setCount ← func to update state count
useState ← function which takes initialState as initial value for any state variable returns 2 values

→ States are mutable
→ reserved for interactivity.

components event handlers may update the state and trigger a UI update

→ state will be set with default value when component mounts and will mutate in time based on user events generated.

Props : allow us to pass data from one component to another component

→ immutable
→ a component cannot change its props however it is responsible for putting together.

<App initial={10} />

In the component App, property would be accessed as this.props.initial

Inorder to access props within the constructor, we need to pass props as an argument to both constructor() and super()

Code:

```
constructor(props){
    super(props)
    console.log(this.props.name)
}
```