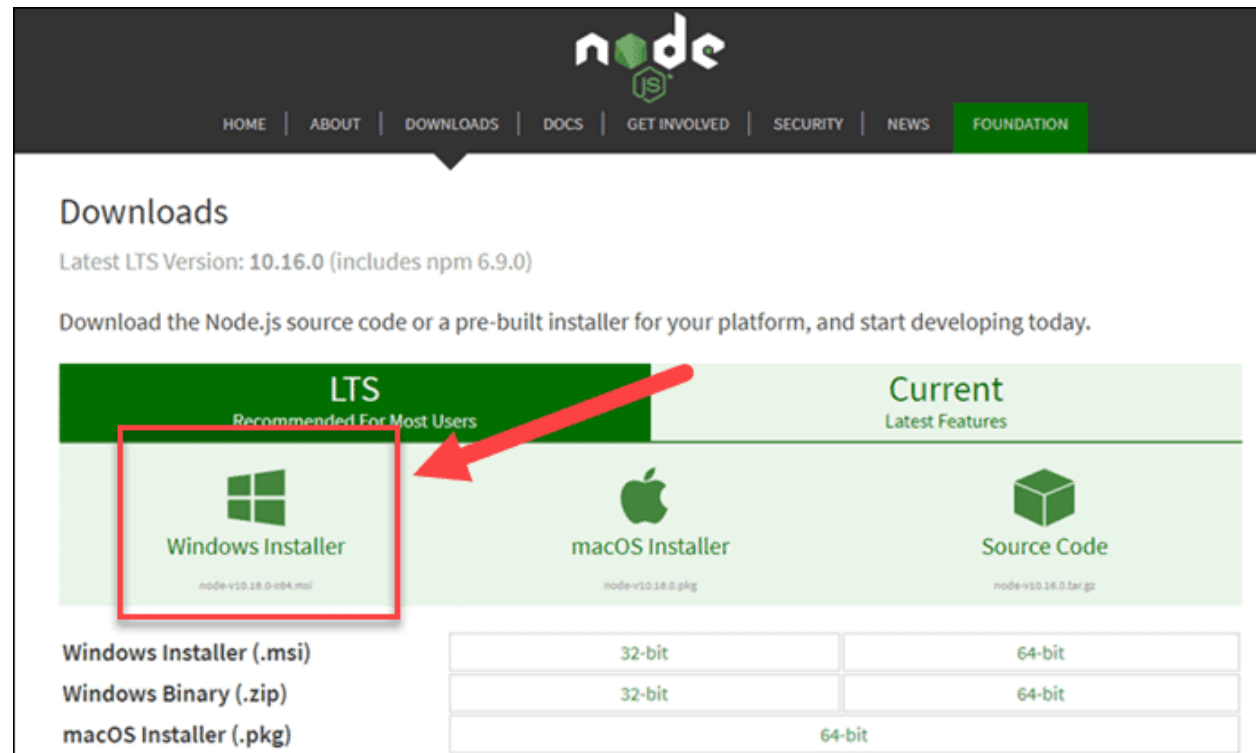# React Fundamentals

1. Node.js
2. NPM
   1. How to install
3. React install
4. Installing libraries
5. Folders and file structure
6. Components
7. Components life cycle
8. State and props

Node.js installation

- **Step 1: Download Node.js Installer**

- In a web browser, navigate to https://nodejs.org/en/download/. Click the **Windows Installer** button to download the latest default version.

- **Step 2: Install Node.js and NPM**
- Once the installer finishes downloading, launch it.

- **Step 3: Verify Installation**
- Open a command prompt (or PowerShell), and enter the following:

- node –v
- npm -v

Node js History:

- In 1995 javascript got introduced and run in web browser through javascript engine and interpreter.
- Mozilla, IE, Chrome, Edge, Safari
- Netscape: Spidermonkey ---- > First JS Engine
- Chrome: V8 ….fast
- In earlier version
- Front End: HTML, CSS, Javascript
- Server End: PHP, ASP.net, JSP
- Ryan Dahl in 2009 thought javascript can run outside the browser as well as script can run on the server side.

NPM

- npm is the world's largest software registry.

- Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.

- npm consists of three distinct components:
  - the website
  - the Command Line Interface (CLI)
  - the registry

React JS

- Generally for building user interfaces react js is used.

- React js is declarative, efficient and flexible library of javascript.

- From MVC (Model, View and Controller) architecture React js acts like a view.

- It is an open source and component based front end library which is responsible only for view purpose.

- It is not a framework. It is only a library.

- It is javascript library for building user interfaces.

Advantages:

- It is user friendly.
- It provides its users with helpful documentation, training resources and in depth tutorials.
- With react js we can build dynamic web application quickly as it offers high functionality.
- It uses JSX(Java Script Extension) that allows you to create an app with minimal coding.
- Reacts components are reusable which makes it easier to develop and maintain your apps.
- Reacts js uses virtual DOM that makes use of in-memory data structure cache and only the final changes are updated in the browser DOM . This makes app faster.
- We can use react on client, server and other framework as well.
- The data and component in react optimize readability which is useful when handling the larger apps.

React Features:

- JSX: JSX supports expression in pure JavaScript syntax
- Components:
  - Core of react
  - Essential when you are handling the code for larger scale projects
- One way data binding:
  - React applies a one way data binding or unidirectional data flow.
  - It gives user a better control over the application.
- Simplicity
  - The JSX file in react simplifies the application.
- Virtual DOM:
  - React creates a virtual DOM i.e. in memory data structure cache.

How React JS works:

- React is a JavaScript library (not a framework) that creates user interfaces in a predictable and efficient way using declarative code.
- You can use it to help build single page applications and mobile apps, or to build complex apps if you utilize it with other libraries.
- Declarative code describes **what** we want instead of saying **how** to do it.
- Declarative code describes the end result, but doesn't act as a step-by-step guide of how to do it.
- In practice, that means declarative code is lightweight, easier to understand and change, and has less bugs.

- ReactJs creates a virtual DOM in memory:
- The virtual DOM stores a representation of the UI in memory and is synced with the actual DOM with the help of React DOM.
- The process of syncing the real DOM with the VDOM is referred to as reconciliation.
- Internally, React uses objects called fibers to keep more details of the component tree.
- The render() function creates a tree of components that is used in the reconciliation process.
- Reconciliation is the process of syncing the VDOM with the real DOM.
- For this to happen, React creates a tree starting from the root node.

- Whenever it encounters root elements that have changed, it tears down the nodes whose states have changed and remounts them.
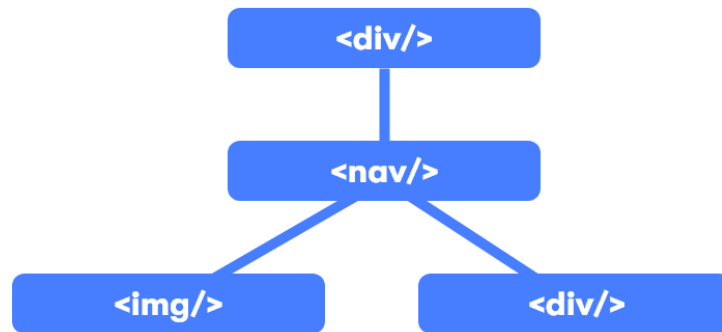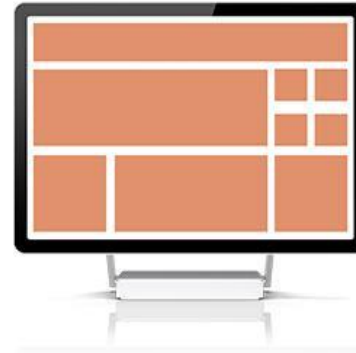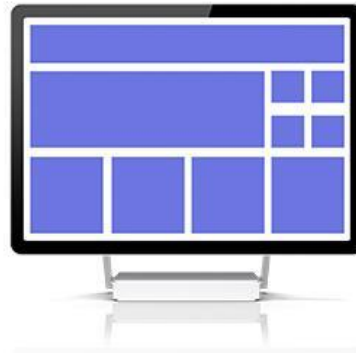

Real DOM
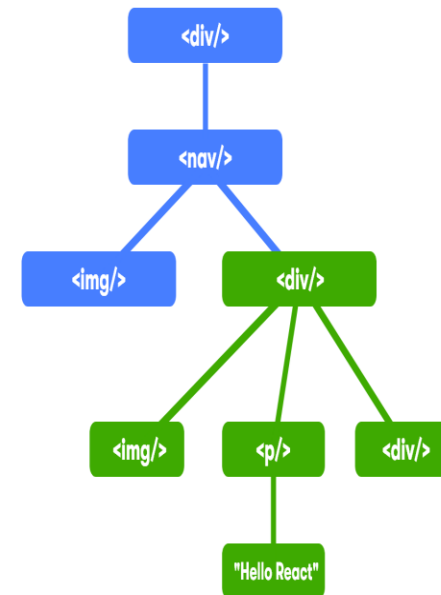
Virtual DOM

React uses Single Page Application(SPA)
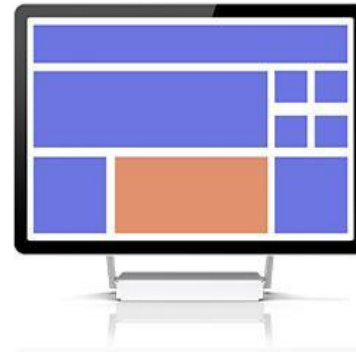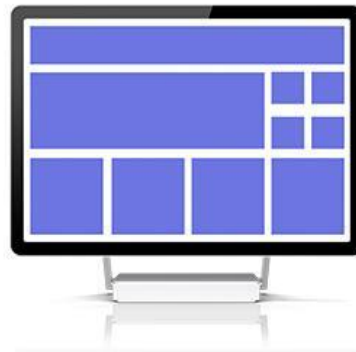
- Time is saved at the server side
- No processing at the server side
- No rendering of pages at the client side
- So overall time is reduced.

- To setup a project in React we must have the following things installed
  - Node.js
  - Visual studio
- After installing the above software traverse to directory when one needs to create the project.
- Open the terminal and execute
- npx create-react-app appname
- Running the above command will create a directory called appname inside the current folder .
- Inside that directory it will generate the initial project structure and install the transitive dependencies.

Project Structure

- README.md

- node_modules

- package.json

- .gitignore

- public

- src
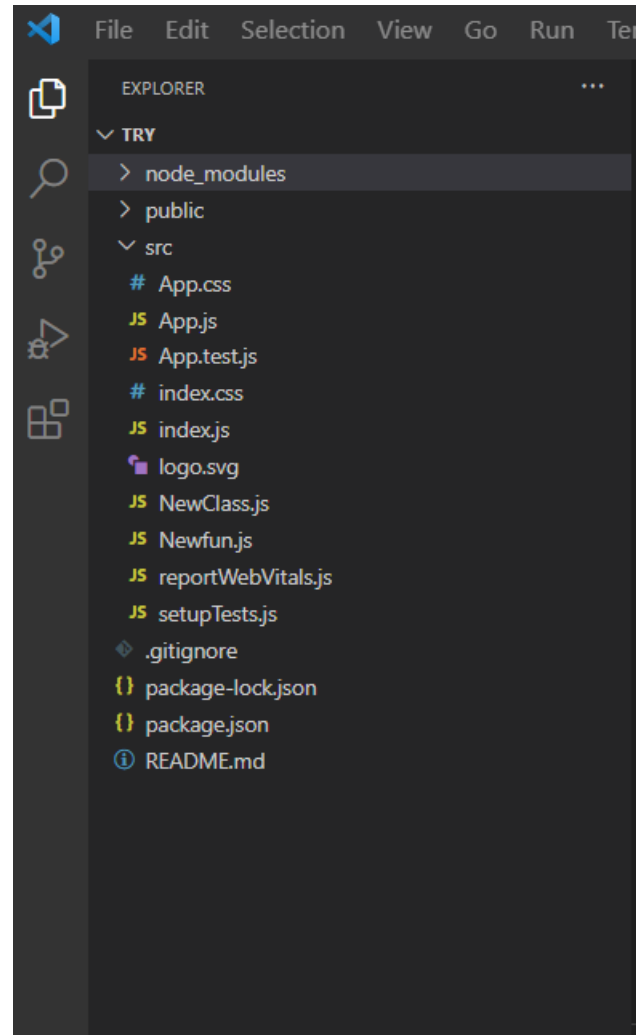
- build represents the path to our final production build.

- This folder would actually be created after we run the npm build.

- We can see all the "dependencies" and "devDependencies" required by our React app in node_modules.

- These are as specified or seen in our package.json file.

- If we just run the ls -l command, we'll see almost 800 sub-directories.

- This directory gets added to .gitignore so it does not really get uploaded/published.

- Also, once we minimize or compress our code for production, our sample app should be less than 100 KB in size.

```
1   ├── README.md
2   ├── node_modules
3   ├── package.json
4   ├── .gitignore
5   ├── build
6   ├── public
7   │   ├── favicon.ico
8   │   ├── index.html
9   │   └── manifest.json
10  └── src
11      ├── App.css
12      ├── App.js
13      ├── App.test.js
14      ├── index.css
15      ├── index.js
16      ├── logo.svg
17      └── serviceWorker.js
```

- Our static files are located in the public directory.
- Files in this directory will retain the same name when deployed to production.
- Thus, they can be cached at the client-side and improve the overall download times.
- All of the dynamic components will be located in the src.
- To ensure that, at the client side, only the most recent version is downloaded and not the cached copy.

App.js

- import React from 'react';
- import logo from './logo.svg';
- import './App.css';
-
  function App() {
-   return (
-     <div className="App">
-       Hello World!!!
-     </div>
-   );
- }
-
  export default App;
-

Index.js

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

index.html

```html
<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
</body>
```

Running App

- In order to run the ReactJS application traverse to the directory where project is created and run

- npm start

- The above command will serve your project on https://localhost:3000

Scripts:

- If we create the app with default template it comes with predefined scripts
  - Start: it helps to run the project locally
  - Test: it helps in running test cases if any for the project
  - Build: it provides optimized production build of the project

Components
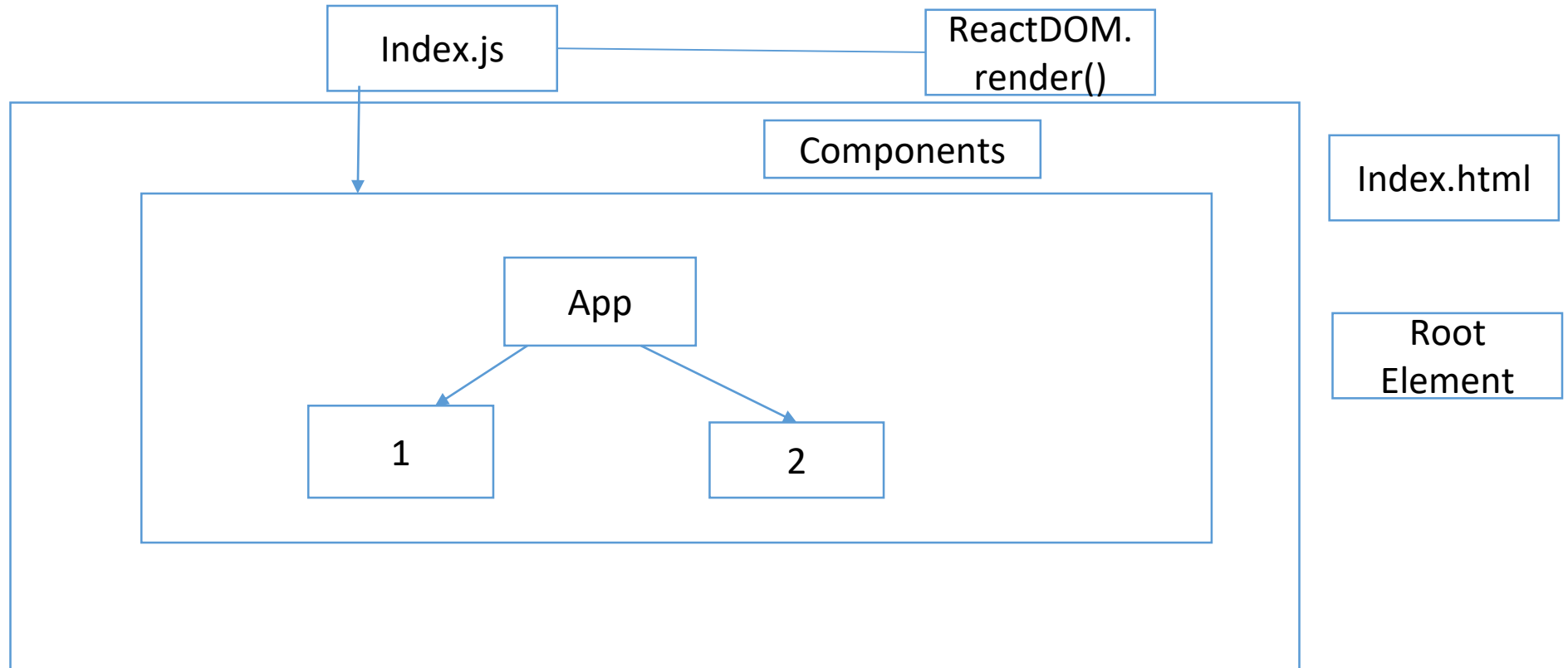
Components:

- Components are the core building blocks of React applications.
- They are logical groups of code that make up the entire application
- With use of component we can build the UIs quickly.
- Every component works independently and we can merge them into a parent component.
- React components have their unique structure, methods and APIs.
- There are two types of components
  - Functional Components
  - Class Components

Component:
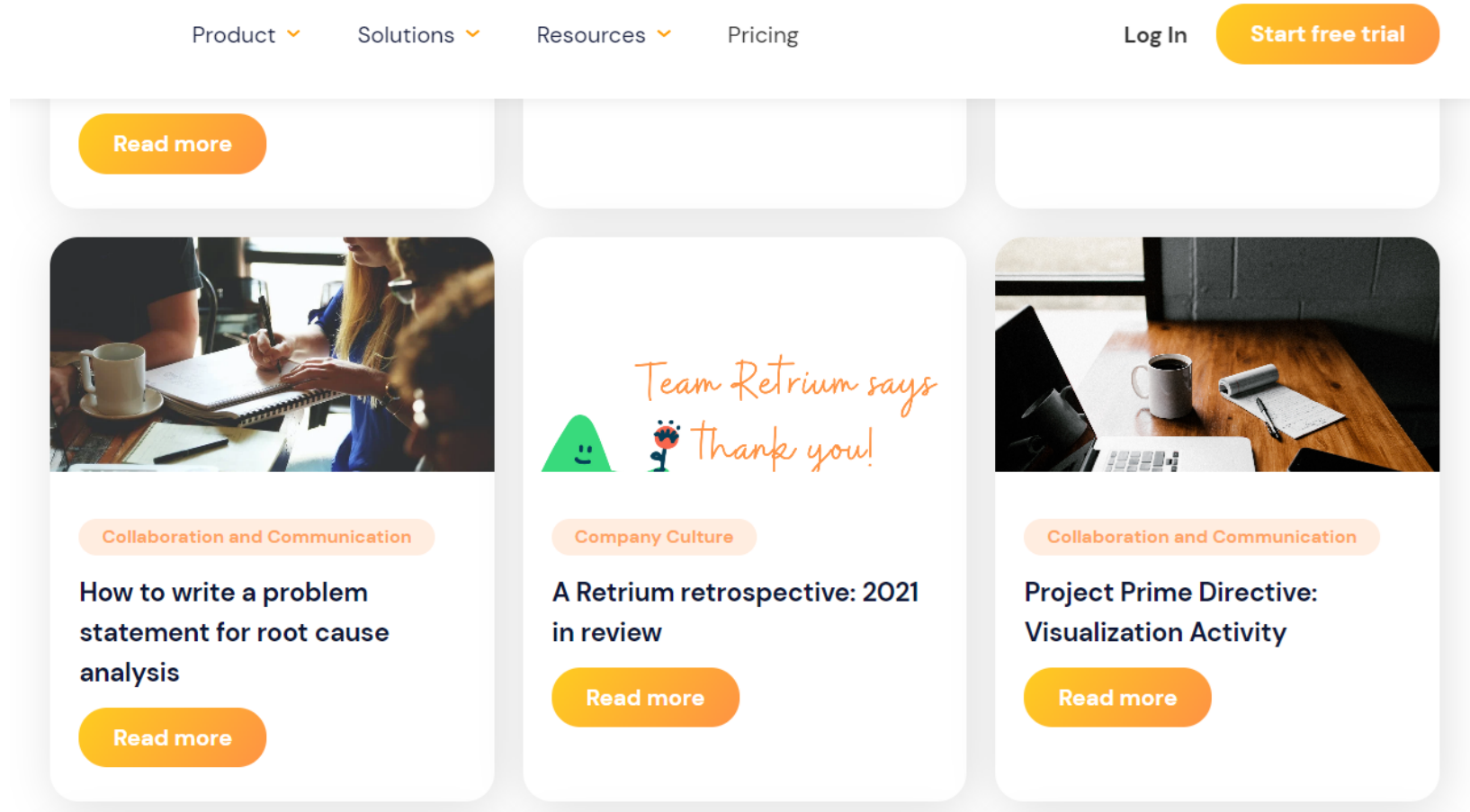
- Components are reusable building blocks in user interface (html and css with js make a component).
- Advantages
  - Reusability
  - Help in maintaining the code

Functional Components

- These are simply JavaScript functions.

- We can create a functional component to React by writing a JavaScript function.

- In the functional Components, the return value is the JSX code to render to the DOM tree.

- There are some benefits you get by using functional components in React.

- Functional components are easier to read and its testable(because these are plain JavaScript functions).

- It's easy to separate container and presentational components

```html
<body>

        <div id="root"></div>

</body>
```

EXPLORER   ...   JS App.js   JS index.js   <> *index.html* ✕

∨ **MYAPP**   public › <> index.html › ⬡ html › ⬡ head › ⬡ meta
  › node_modules   9   name="description"
 ∨ public   10   content="Web site created using create-react-app"
    ⭐ favicon.ico   11   />
    <> index.html   12   <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    🖼 logo192.png   13   <!--
    🖼 logo512.png   14   manifest.json provides metadata used when your web app is installed on a
    {} manifest.json   15   user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    ☰ robots.txt   16   -->
  ∨ src   17   <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    # App.css   18   <!--
    JS App.js   19   Notice the use of %PUBLIC_URL% in the tags above.
    JS App.test.js   20   It will be replaced with the URL of the `public` folder during the build.
    # index.css   21   Only files inside the `public` folder can be referenced from the HTML.
    JS index.js   22
    🔷 logo.svg   23   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    JS reportWebVitals.js   24   work correctly both with client-side routing and a non-root public URL.
    JS setupTests.js   25   Learn how to configure a non-root public URL by running `npm run build`.
  ◆ .gitignore   26   -->
  {} package-lock.json   27   <title>React App</title>
  {} package.json   28   </head>
  ⓘ README.md   29   <body>
                                            30   <noscript>You need to enable JavaScript to run this app.</noscript>
                                            31   <div id="root"></div>
                                            32   <!--
                                            33   This HTML file is a template.
                                            34   If you open it directly in the browser, you will see an empty page.
                                            35

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <h1>Hello TE IT</h1>
);
```
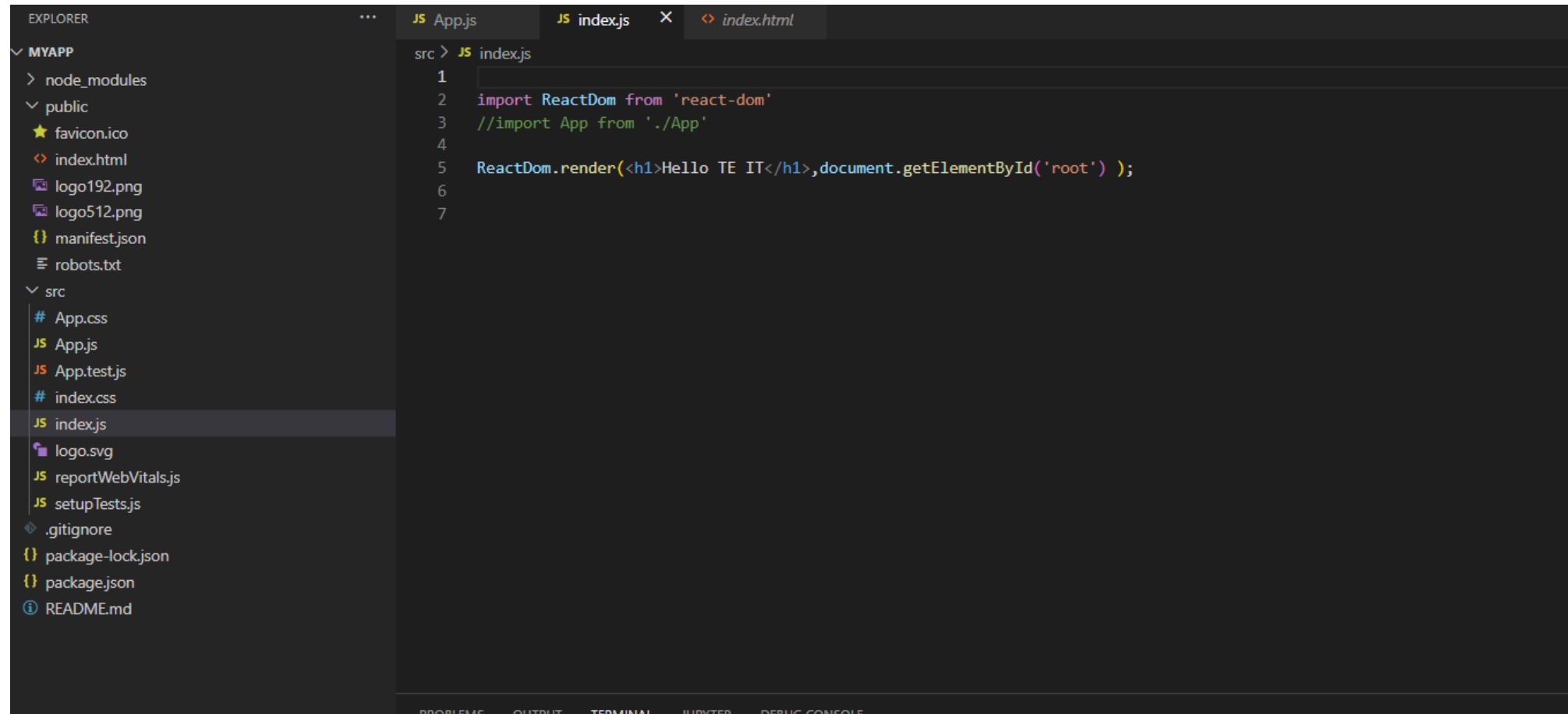
Hello TE IT

Functional Component (Home.js)

```javascript
import React from 'react'
export default function Home()
{
    return(
      <div>Hello from functional component</div>
    )
}
```

```
import logo from './logo.svg';
import './App.css';
import Home from './Home';
import React from 'react'
//import Prof from './Prof'
function App() {
  return (
    <div>
    <Home />
    </div>
    )
 }
export default App
```

Functional Component:

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

- If you want to pass more than one tag in same return statement then always add it into wrapper.

- Try to create separate css for separate component

Class Name

```
return (
   <div className="App">
     Hello World!!! First Component
   </div>
);
```

Class Components

- These components are simple classes (made up of multiple functions that add functionality to the application).

- All class based components are child classes for the Component class of ReactJS.

User.js

```
import React, {Component} from 'react'
export default class User extends Component
{
  render()
  {
    return(
    <h1>Hello from class component</h1>
    )
  }
}
```

```
App.js

import React from 'react';
import logo from './logo.svg';
import './App.css';
import User from './User'


function App() {
  return (
    <div className="App">
      Hello World!!! First Component
      <User />
    </div>
  );
}


export default App;
```
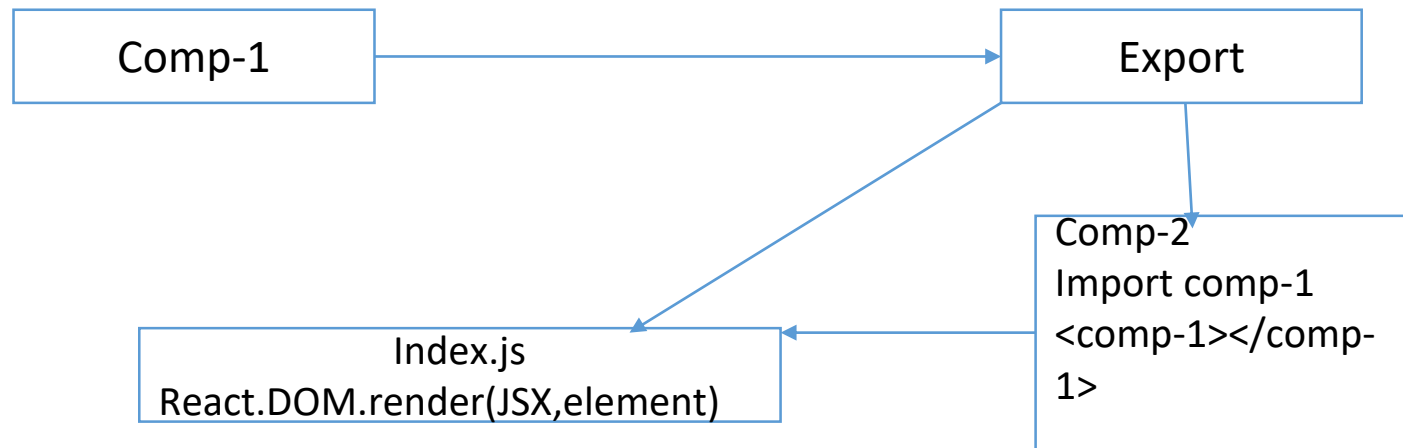
Hello World!!! First Component

Hello from class component

```
┌─────────────────┐                                      ┌─────────────────┐
│     Comp-1      │ ───────────────────────────────────▶ │     Export      │
└─────────────────┘                                      └─────────────────┘
                                          ╲                        │
                                           ╲                       ▼
                                            ╲            ┌─────────────────────┐
                                             ╲           │ Comp-2              │
                                              ╲          │ Import comp-1       │
┌─────────────────────────────┐               ╲         │ <comp-1></comp-     │
│ Index.js                    │ ◀──────────────╲────────│ 1>                  │
│ React.DOM.render(JSX,element)│                         │                     │
└─────────────────────────────┘                         └─────────────────────┘
```

Functional Vs Class Component

- Functional Component:
  - Simple functions
  - Use functional component as much as possible
  - Solution without using state
  - Absence of this keyword
  - Stateless/dumb/Presentational Components
- Class component
  - More features
  - Maintain own private data called as states
  - Complex UI logic
  - Provide life cycle methods
  - Stateful/ Smart/ Container

```jsx
import React from 'react'
export default function Home()
{

    const f1=()=>
    {
        alert("alert from functional component");
    }
    return <div>
      <h1 onClick={f1}>Hello from functional component</h1>
    </div>
}
```

ReactJS JSX

- JSX is a JavaScript Extension Syntax used in React to easily write HTML and JavaScript together.

- E.g const h1tag="<h1> Hello</h1>"

- JSX produces React element.

- Embedding expression in JSX
  ```
  ReactDOM.render(
  Element,
  Document.getElementById('root');
  );
  ```

Why JSX:

- In React we can write HTML/XML structures in the same file as the javascript code.
- Then the preprocessor converts the expression into javascript code.
- JSX tags also includes a tag name, attributes and children.
- Reasons to use JSX in react are
  - It works at a more incredible speed than regular javascript.
  - While it translates the code to javascript it also performs optimization.
  - It makes use of components that include both markup and logic.
  - We can find the errors during compilation time as it is type safe.
  - We can easily create template in JSX.

ReactJS Props and State

```
function add(p,q)
{
console.log(p);
console.log(q);
}
function pass()
{
a=10;
b=20;
add(a,b)
}
```

```
function add1(p,q)
{
p=p+10;
q=q+20;
m=p+q;
console.log(m);
}

function pass()
{
a=10;
b=20;
add1(a,b)
}
```

ReactJS Props:

- Props stand for "Properties."
- They are read-only components.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It gives a way to pass data from one component to other components.
- It is similar to function arguments.
- Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component.
- Inside the components, we can add attributes called props.
- These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

ReactJS Props:

- The components receives the arguments as props object in function as well as class component.

- To send props into a component use the same syntax as HTML attributes/ property.

-

- <student name="abc"> </student>

- <student name="abc" />

App.js

```
function App() {
  return (

    <div>

    <Home text="Hello Props"/>

    </div>

    )

}
```

Home.js

```
export default function Home(props)
{
    return <div>
      <h1>{props.text}</h1>
    </div>
}
```

```
function App() {
  return (
    <div>
      <Home text={{name:"abc"}}/>
    </div>
    )

}
```

```
export default function
Home(props)
{
    return <div>
     <h1>{props.text.name}</h1>
    </div>
}
```

```
function App() {
  return (
    <div>
    <Home text={{name:"abc"}}
data="Profile data"/>
    </div>
    )

}
```

```
export default function Home(props)
{
    return <div>
      <h1>{props.text.name}</h1>
      <h2>{props.data}</h2>
    </div>
}
```

```
Student.js

import React from 'react'

export default class Student extends
React.Component{
    render(){
        return(
            <div
style={{backgroundColor:'skyblue', margin:20}}>
            <h1>  Hello {this.props.name}</h1>
                from {this.props.class}
            </div>
        )
    }
}
```

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import Student from './Student';
//import User from './User'


function App() {
    return (
        <div className="App">
            Props
        <Student name="Ramesh" class="BE"/>
        <Student name="Suresh" class="BE"/>
        </div>
    );
}


export default App;
```

Props

Hello Ramesh

from BE

Hello Suresh

from BE

ReactJS Props:

- ReactDOM.render(<student name="abc" />
- <student name="abc" class="BE" />

Student=
{
Name:"abc",
Class:"BE"
}
student.name, student. class

ReactJS Props:

```
function student(props){
return <h1> Hello, {props.name}</h1>


Class student extends React.component
{
render(){
        return(<h1>Hello, {this.props.name}</h1>);
}
}
```
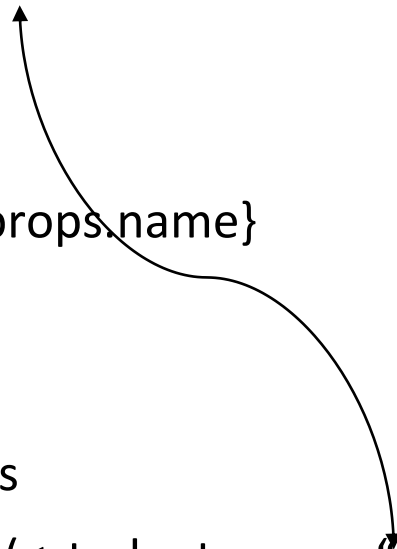
ReactJS Props:

function student(props){

return(

<div>

<h1>your name,{props.name}

</div>)

}

JSX------- > index.js

ReactDOM.render(<student name="abc" />, document.getElementById("root"));

ReactDOM.render(<student name="{abc}" />, document.getElementById("root"));

ReactDOM.render(<student name="{abc}" rollno={1} />, document.getElementById("root"));

ReactJS Props:

```
class student extends React.component
{render(){
return(
<div>
<h1>your name,{this.props.name}
</div>)
}
JSX-------  > index.js
ReactDOM.render(<student name="abc" />, document.getElementById("root"));
ReactDOM.render(<student name="{abc}" />, document.getElementById("root"));
```

ReactJS State

- The state is a built-in React object that is used to contain data or information about the component.

- A component's state can change over time;

- Whenever it changes, the component re-renders.

- The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.

- State object is used only in class component
- Without constructor
- Inside the constructor
- We can modify the state using setState object

- In ReactJs the state contains information about the component and can change over time.

- It can be either a response to user action or a system event.

- Data collected in a state is a private object.

- Component within the state are called stateful components.

```
import React from 'react';
import Student from './Student';

class App extends React.Component {
 constructor()
 {
      super();
      this.state={
      name:"Ramesh"
}}
  render()
  {
    return (
      <div className="App">
       Props
      <Student name={this.state.name} class="BE"/>
     <button onClick={()=>this.setState({name:"Rupali"})}>Update Name
</button>        </div>
    );
  }}
export default App;
```

```jsx
export default class Home extends
React.Component
{
    constructor()
    {
        super();
        this.state={
            name:'Nikhil',
            email:'nikhil@test.com',
            count:0
        }
    }
    Updatestate()
    {
        this.setState({
            name:'Rupali',
            count:this.state.count+1
        })
    }

    render()
    {
        return(
            <div>

<h1>Hello{this.state.count}</h1>

<h3>Email:{this.state.email}</h3>
                <button
onClick={()=>{this.Updatestate()}}>Update
State</button>

            </div>
        )
    }
}
```

ReactJS Component Life Cycle

- React web apps are actually a collection of independent components that run according to the interactions made with them.

- Every React Component has a lifecycle of its own, lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence.

- A React Component can go through four stages of its life
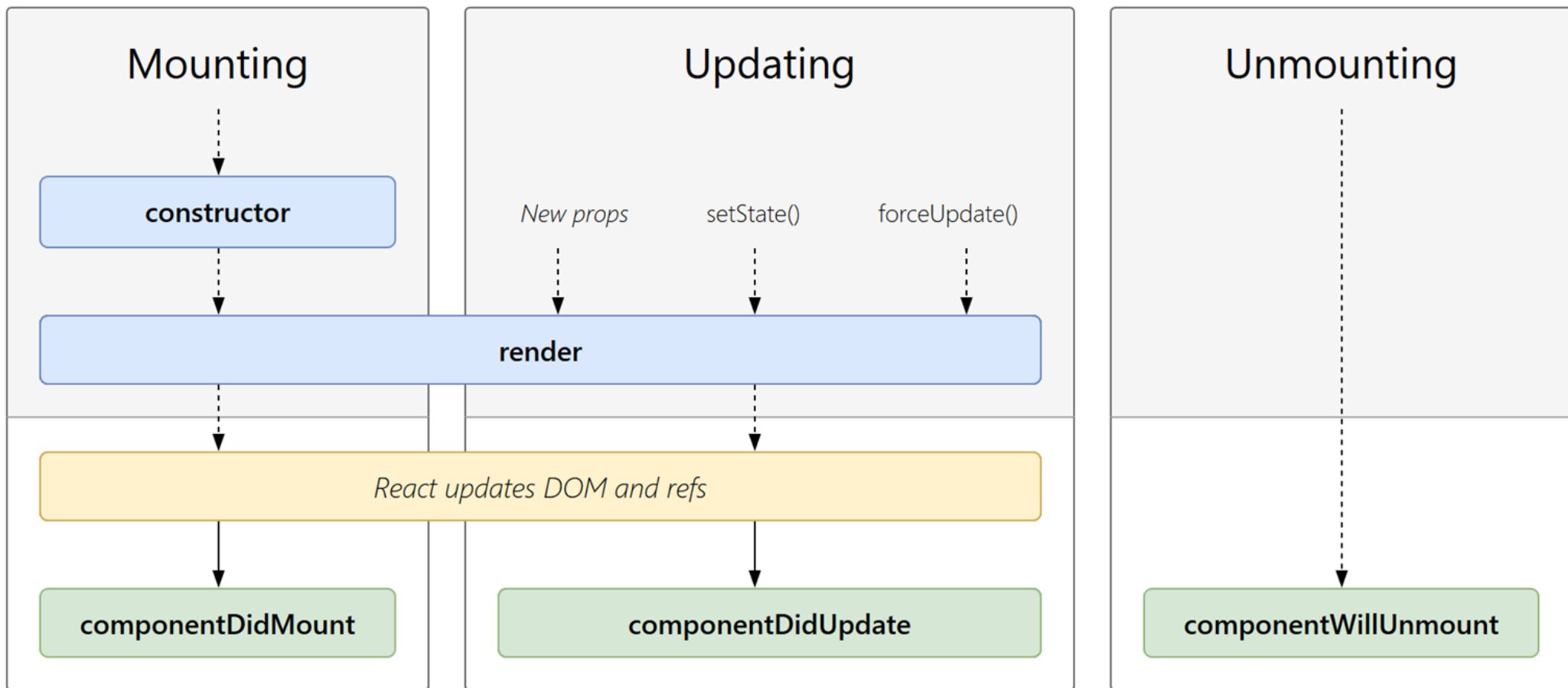
```
class App extends React.Component {
 constructor()
 {
super();
console.warn("Constructor")
}
 componentDidMount()
 {
  console.warn("componentDidMount")
 }

  render()
  {
    console.warn("render")
    return (--------
```

Warning: ReactDOM.render is no longer   react-dom.development.js:86
supported in React 18. Use createRoot instead. Until you switch to the new
API, your app will behave as if it's running React 17. Learn more: https://r
eactjs.org/link/switch-to-createroot

⚠ ▸ Constructor   App.js:13
⚠ ▸ render   App.js:22
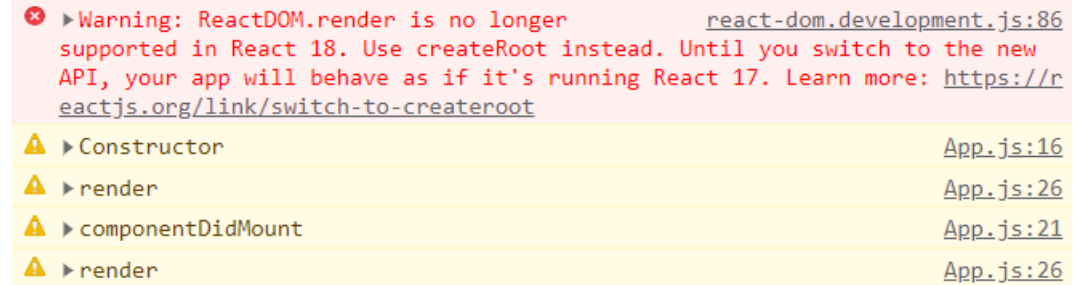⚠ ▸ componentDidMount   App.js:17

Initial Phase:

- It is the birth phase of the lifecycle of a ReactJS component.
- Here, the component starts its journey on a way to the DOM.
- 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.

Mounting:

- In this phase, the instance of a component is created and inserted into the DOM.
- Constructor()
- It consists of the following methods.
- getDerivedStateFromProps()   // status
- render()
    - This method is defined in each and every component.
    - It is responsible for returning a single root HTML node element.
    - If you don't want to render anything, you can return a null or false value.
- componentDidMount()
    - This is invoked immediately after a component gets rendered and placed on the DOM.
    - Now, you can do any DOM querying operations.
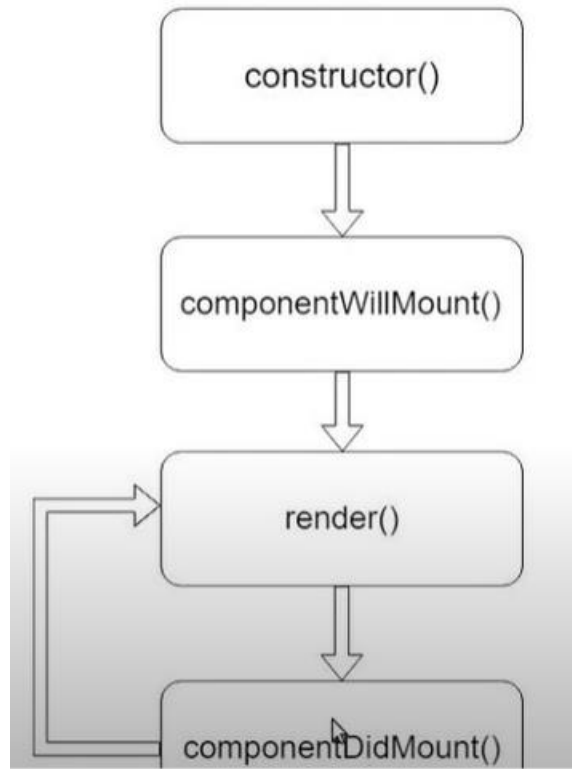
```
class App extends React.Component {
 constructor()
 {
super();
this.state={
  data:null
}
console.warn("Constructor")
}
 componentDidMount()
 {
  this.setState({data:"updated"})
  console.warn("componentDidMount")
 }
 render()
 {
   console.warn("render")
   return (
     <div>
        <h1>Hi</h1>
        </div>
     );
  }}export default App;
```

```
⊗  ▶Warning: ReactDOM.render is no longer          react-dom.development.js:86
   supported in React 18. Use createRoot instead. Until you switch to the new
   API, your app will behave as if it's running React 17. Learn more: https://r
   eactjs.org/link/switch-to-createroot
⚠  ▶Constructor                                                    App.js:16
⚠  ▶render                                                         App.js:26
⚠  ▶componentDidMount                                              App.js:21
⚠  ▶render                                                         App.js:26
```

- Updating
- The third stage starts when the component has been adopted on the browser. This can then grow by receiving new updates from the program. The user can interact with the program, and then the component can be updated according
- getDerivedStateFromProps()
- shouldComponentUpdate() //permission to view
- The method tells the program about the state of rendering when it is updated.
- If new props or rules are being updated, then a rendering can be done or skipped.
- Updating the method as true/false is the proper approach.
- The default here is true, which can be changed as per the code.

- getSnapShotBeforeUpdate()
- componentDidUpdate – This is then executed when the updated component has been updated in the DOM as well.
- You can then initiate new libraries to reload as well so that you can maintain an updated program throughout the process.

```
class App extends React.Component {
 constructor()
 {
super();
this.state={
active:null,
who:null
}}
 componentDidUpdate()
 {
  console.warn("componentDidMount")
 }
 render()
  {
    console.warn("render")
    return (
      <div>
        <h1>Hi</h1>
        <button
onClick={()=>this.setState({active:"update"})}>
Update</button>
        </div>
    );}}
```

```
class App extends React.Component {
 constructor()
 {
super();
this.state={
active:null,
who:null
}
componentDidUpdate()
 {
  console.warn("componentDidMount")
  this.setState({who:"Mahesh"})
 }
  render()
  {
    console.warn("render")
    return (
      <div>
      <button
onClick={()=>this.setState({active:"update"})}>
Update</button>
        </div>
    );}
```

▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27
▶ componentDidMount	App.js:21
▶ render	App.js:27

```
class App extends React.Component {
 constructor()
 {
super();
this.state={
active:null,
who:null
}}
 componentDidUpdate()
 {
  console.warn("componentDidMount")
  if(this.state.who==null)
  {
  this.setState({who:"Mahesh"})
 }}
 render()
 {
   console.warn("render")
   return (
     <div>
        <button onClick={()=>this.setState({active:"update"})}>Update</button>
     </div>
   );}}
export default App;
```

- Unmounting:
- The final stage of unmounting is essential as it doesn't require the component and gets unmounted from the DOM.
- As the final state, it is designed to produce the outcome via unmounting.
- componentWillUnmount – This is the last method in the lifecycle as it pertains to the core unmounting and removal from the DOM.
- The cleaning up of the component is also performed here.
- This is also used in the logging out of users when they want to clear out the program from their browser.

- componentWillUnmount() is invoked immediately before a component is unmounted and destroyed.

- You should not call setState() in componentWillUnmount() because the component will never be re-rendered.

- Once a component instance is unmounted, it will never be mounted again.

ReactJS Form:

- Form 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.

ReactJS Router:

- Routing is a process in which a user is directed to different pages based on their action or request.
- ReactJS Router is mainly used for developing Single Page Web Applications.
- React Router is used to define multiple routes in the application.
- When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.
- React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package.
- It provides the synchronous URL on the browser with data that will be displayed on the web page.
- It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

- React Router plays an important role to display multiple views in a single page application.

- Without React Router, it is not possible to display multiple views in React applications.

- Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

Components in React Router:

- There are two types of router components:
- **<BrowserRouter>:** It is used for handling the dynamic URL.
- **<HashRouter>:** It is used for handling the static request.

Setup Routing for an App

- Step 1:
- Install a React Router:
- npm install react-router-dom
- Step 2:
- Create components
- Step 3:
- Add a Router

About.js

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

App.js

```
import React from 'react'
class App extends React.Component {
  render() {
    return (
      <div>
       <h1>Home</h1>
      </div>
    )
  }
}
export default App
```

Contact.js

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

- Step-2: For Routing, open the index.js file and import all the three component files in it.

- import { Route, Link, BrowserRouter as Router } from 'react-router-dom' which helps us to implement the Routing.

- Route is used to define and render component based on the specified path.

- It will accept components and render to define what should be rendered.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Adding Navigation using Link component

- When we click on any of that particular Link, it should load that page which is associated with that path without reloading the web page.

- To do this, we need to import <Link> component in the index.js file.

- This component is used to create links which allow to navigate on different URLs and render its content without reloading the webpage.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from
'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <Link to="/">Home</Link>
        </li>
        <li>
          <Link to="/about">About</Link>
        </li>
        <li>
          <Link to="/contact">Contact</Link>
        </li>
      </ul>

      <Route exact path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)
ReactDOM.render(routing,
document.getElementById('root'));
```

```
function App()
{
  return(
    <div>
      <h1>React Forms</h1>
      <form>
        <input type="text" placeholder= "name"></input><br/><br/>
        <select>
          <option>select option</option>
          <option>Coffee</option>
          <option>tea</option>
        </select>
        <input type="checkbox" /><span>Accept terms and conditions</span><br/><br/>
        <input type="submit">Submit</input>
            </form>
    </div>
  )}
  export default App;
```

```
function App()
{
 function getformdata(e)
 {
   e.preventDefault()
 }
   return(
     <div>
        <h1>React Forms</h1>
        <form>
          <input type="text" placeholder= "name"></input><br/><br/>
          <select>
            <option>select option</option>
            <option>Coffee</option>
            <option>tea</option>
          </select>
          <input type="checkbox" /><span>Accept terms and conditions</span><br/><br/>
          <input type="submit">Submit</input>
                </form>
      </div>
   )}
   export default App;
```

React Events

- An event is an action that could be triggered as a result of the user action or system generated event.

- E.g a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

- 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.

- In react, we cannot return false to prevent the default behavior.

- We must call preventDefault event explicitly to prevent the default behavior

ReactJS Events

Handling events with React elements is very similar to handling events on DOM elements.

There are some syntax differences:

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

```
<button onclick="f1()">
 Hello
</button>
```

```
<button onClick={f1}>
 f1
</button>
```

Another difference is that you cannot return false to prevent default behavior in React.

You must call preventDefault explicitly.

e.g to prevent the default form behavior of submitting, you can write:

```
<form onsubmit="console.log('You clicked submit.'); return false">
  <button type="submit">Submit</button>
</form>
```

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log('You clicked submit.');
  }
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```
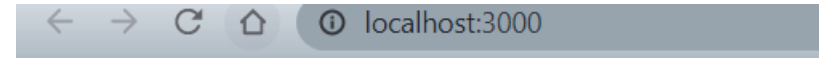
React CSS

- CSS in React is used to style the React App or Component.
- The **style** attribute is the most used attribute for styling in React applications, which adds dynamically-computed styles at render time.
- Inline Styling:
  - The inline styles are specified with a JavaScript object

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "Blue"}}>Hello TE IT</h1>
      <p>Welcome to Internet Programming</p>
      </div>
    );
  }
}
export default App;
```


In JSX, JavaScript expressions are written inside curly braces, and JavaScript objects also use curly braces,

camelCase Property Name

- if the properties have two names, it must be written in camel case syntax.

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "Red"}}>TE IT</h1>
      <p style={{backgroundColor: "lightgreen"}}>Welcome to
Internet Programming</p>
      </div>
    );
  }
}
export default App;
```
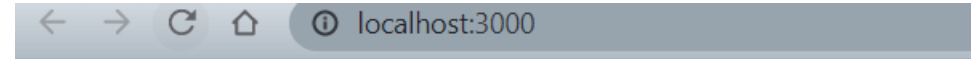
Using Javascript Object:

The inline styling also allows us to create an object with styling information and refer it in the style attribute.

```
import React from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component {
  render() {
    const mystyle = {
      color: "Green",
      backgroundColor: "lightBlue",
      padding: "10px",
      fontFamily: "Arial"
    };
    return (
      <div>
      <h1 style={mystyle}>TE IT</h1>
      <p style={{backgroundColor: "lightgreen"}}>Welcome to Internet Programming</p>
      </div>
    );
  }
}
export default App;
```

CSS Style Sheet

- We can write styling in a separate file for React application, and save the file with a .css extension.

- We you can **import** this file in our application.

```
App.js
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1>TE IT</h1>
      <p>Welcome to Internet Programming</p>
      </div>
    );  }  }
export default App;
```

```
App.css
body {
  background-color: #008080;
  color: yellow;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

TE IT

Welcome to Internet Programming

CSS Module

- CSS Module is another way of adding styles to the application.
- It is a CSS file where all class names and animation names are scoped locally by default.
- It is available only for the component which imports it, means any styling we add can never be applied to other components without our permission.
- We can create CSS Module with the .module.css extension like a myStyles.module.css name.

```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import style from './myStyles.modules.css';

class App extends React.Component {
  render() {
    return (
      <div>
      <h1 className={style.mystyle}>TE
IT</h1>
      <p className={style.parastyle}>Welcome to
Internet programming.</p>
      </div>
    );
  }
}
export default App;
```

```css
myStyle.module.css

.mystyle {
    background-color: #cdc0b0;
    color: Red;
    padding: 10px;
    font-family: Arial;
    text-align: center;
}


.parastyle{
    color: Green;
    font-family: Arial;
    font-size: 35px;
    text-align: center;
}
```

Styled Component

- Styled-components is a library for React.

- It uses enhance CSS for styling React component systems in the application, which is written with a mixture of JavaScript and CSS.

- The styled-components provides:
  - Automatic critical CSS
  - No class name bugs
  - Easier deletion of CSS
  - Simple dynamic styling
  - Painless maintenance

Installation

- The styled-components library takes a single command to install in React application.

- $ npm install styled-components --save

```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import styled from 'styled-components';
class App extends React.Component {
  render() {
    const Div = styled.div`
          margin: 20px;
          border: 5px dashed green;
          &:hover {
          background-color: ${(props) => props.hoverColor};
          }
          `;
    const Title = styled.h1`
          font-family: Arial;
          font-size: 35px;
          text-align: center;
          color: palevioletred;
          `;
    const Paragraph = styled.p`
          font-size: 25px;
          text-align: center;
          background-Color: lightgreen;
          `;
                                              return (
                                                   <div>
                                                        <Title>Styled Components Example</Title>
                                                        <p></p>
                                                        <Div hoverColor="Orange">
                                                             <Paragraph>Hello TE IT</Paragraph>
                                                        </Div>
                                                   </div>
                                              );
                                           }
                                        }
                                        export default App;
```
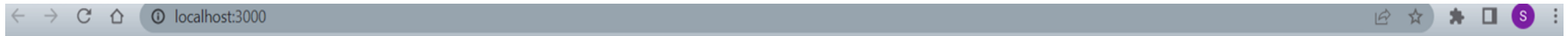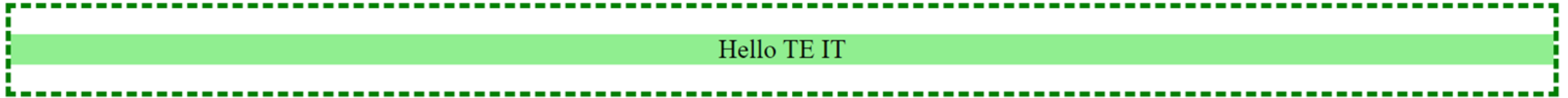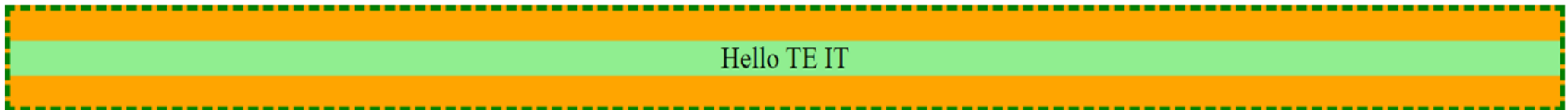
ReactJS Animation

- we can add animation using an explicit group of components known as the **React Transition Group**.

- React Transition Group is an add-on component for managing component states and useful for
defining **entering** and **exiting** transitions.

- It is not able to animate styles by itself.

- Instead, it exposes transition states, manages classes and group elements, and manipulates the DOM in useful ways.

- It makes the implementation of visual transitions much easier.

- We need to install **react-transition-group** for creating animation in React Web application.
- npm install react-transition-group

React Transition Group Components:

- React Transition Group API provides **three** main components.

- These are:

  1. Transition
  2. CSSTransition
  3. Transition Group

Transition

- It has a simple component API to describe a transition from one component state to another over time.
- It is mainly used to animate the **mounting** and **unmounting** of a component.
- It can also be used for in-place transition states as well.
- We can access the Transition component into four states:
- entering
- entered
- exiting
- exited

CSSTransition

- The CSSTransition component uses CSS stylesheet classes to write the transition and create animations.
- It can also inherit all the props of the transition component.
- We can divide the "CSSTransition" into **three** states.
- These are:
1. Appear
2. Enter
3. Exit
- CSSTransition component must be applied in a pair of class names to the child components.
- The first class is in the form of **name-stage** and the second class is in the **name-stage-active**

TransitionGroup

- This component is used to manage a set of transition components (Transition and CSSTransition) in a list.

- It is a state machine that controls the **mounting** and **unmounting** of components over time.

- The Transition component does not define any animation directly.

- It means, the "TransitionGroup" component can have different animation within a component.

ReactJS Forms

- HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state.
- In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.
- In React, mutable state is typically kept in the state property of components, and only updated with setState().

The React component that renders a form also controls what happens in that form on subsequent user input.

An input form element whose value is controlled by React in this way is called a "controlled component".

```jsx
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange =
this.handleChange.bind(this);
    this.handleSubmit =
this.handleSubmit.bind(this);
  }
 handleChange(event) {
    this.setState({value:
event.target.value});
 }
  handleSubmit(event) {
    alert('A name was submitted: ' +
this.state.value);
    event.preventDefault();
  }
```

```jsx
  render() {
return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text"
value={this.state.value}
onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit"
/>
      </form>
    );
  }
}
```

The select Tag

```
<select>
  <option value="grapefruit">Grapefruit</option>
  <option value="lime">Lime</option>
  <option selected value="coconut">Coconut</option>
  <option value="mango">Mango</option>
</select>
```

- Coconut option is initially selected, because of the selected attribute.
- React, instead of using this selected attribute, uses a value attribute on the root select tag.

```jsx
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value: event.target.value});
}

  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Pick your favorite flavor:
          <select value={this.state.value}
onChange={this.handleChange}>
<option value="grapefruit">Grapefruit</option>
<option value="lime">Lime</option>
<option value="coconut">Coconut</option>
<option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```