**Week 5**

**Introduction to React**

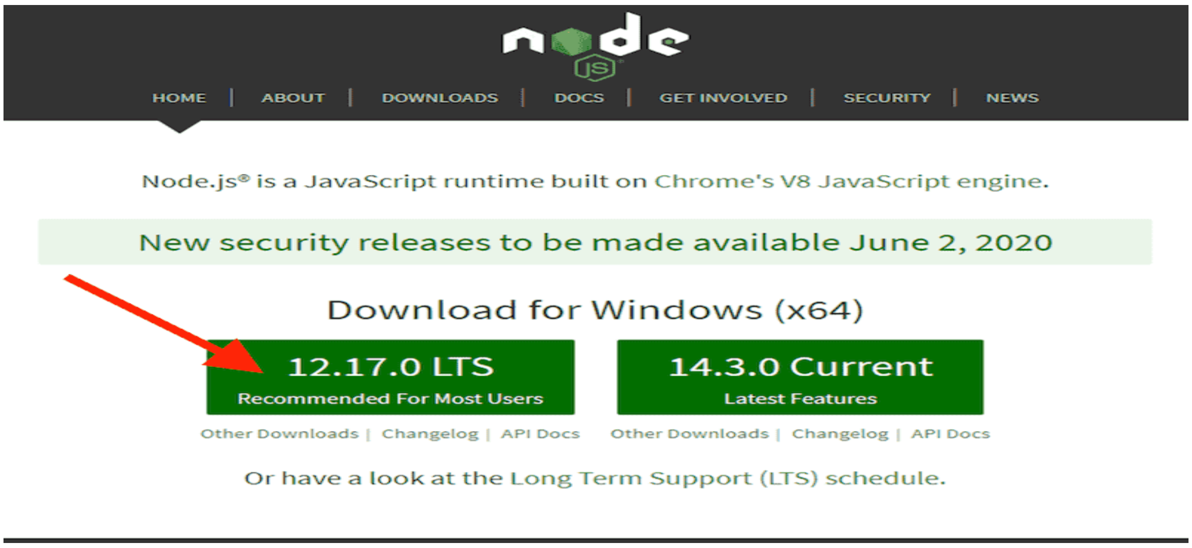
React is an open source frontend JavaScript library for building complex UIs; it allows the programmers to create complex UIs from smaller components.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

**Setting up React development**

Step 1: Install NodeJS.

Download Node.js and npm from https://nodejs.org.



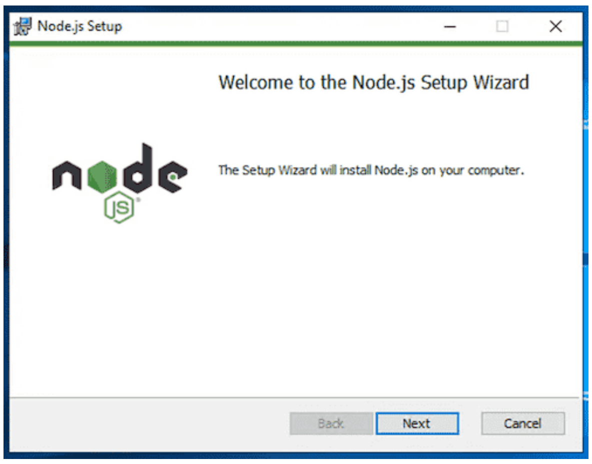
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

New security releases to be made available June 2, 2020

Download for Windows (x64)

HOME
ABOUT
DOWNLOADS
DOCS
GET INVOLVED
SECURITY
NEWS
12.17.0 LTS
Recommended For Most Users
14.3.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

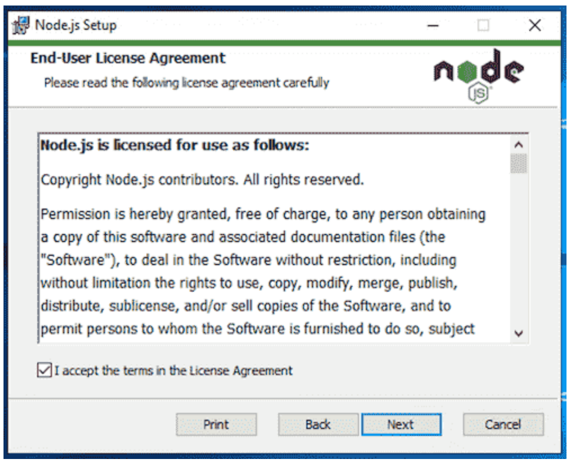
Or have a look at the Long Term Support (LTS) schedule.



Node.js Setup

Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.



Node.js Setup

End-User License Agreement

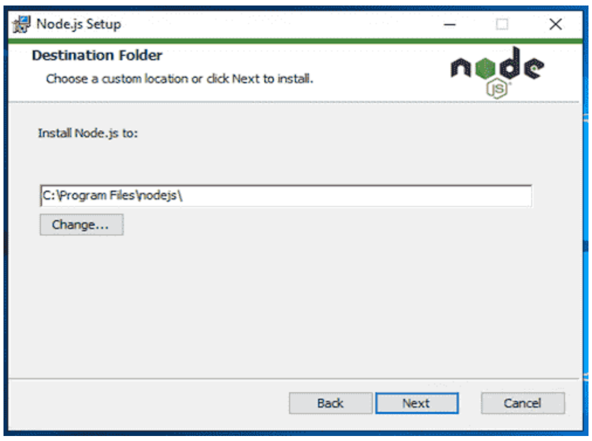
Please read the following license agreement carefully

Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

[✓] I accept the terms in the License Agreement

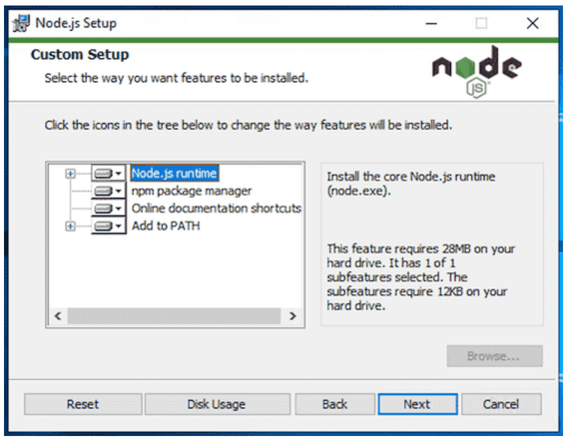

Node.js Setup

**Destination Folder**

Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\


Node.js Setup

**Custom Setup**

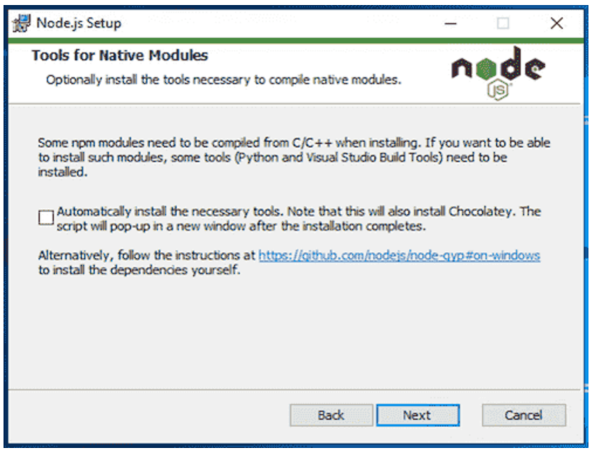
Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH

Install the core Node.js runtime (node.exe).

This feature requires 28MB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 12KB on your hard drive.


Node.js Setup

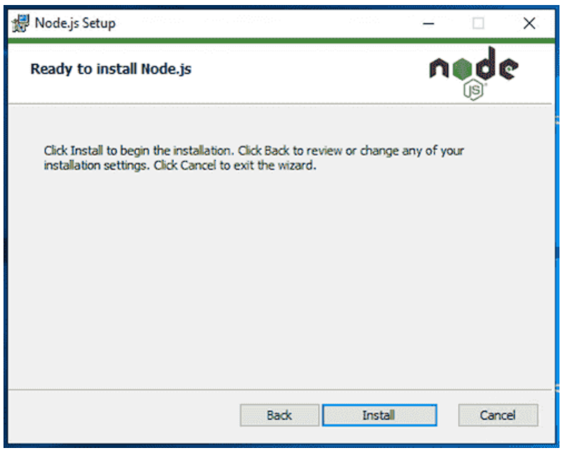
**Tools for Native Modules**

Optionally install the tools necessary to compile native modules.

Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.

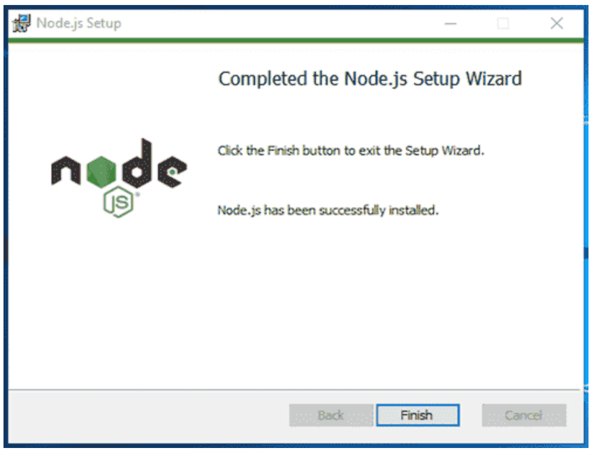
☐ Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.

Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.


Node.js Setup

**Ready to install Node.js**

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.


Node.js Setup

Completed the Node.js Setup Wizard

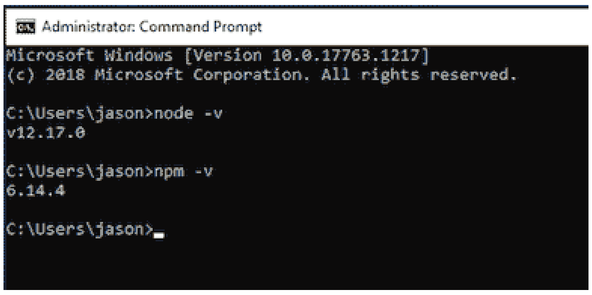
Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

Test that Node.js and npm were installed successfully by running the commands

node -v

npm -v.

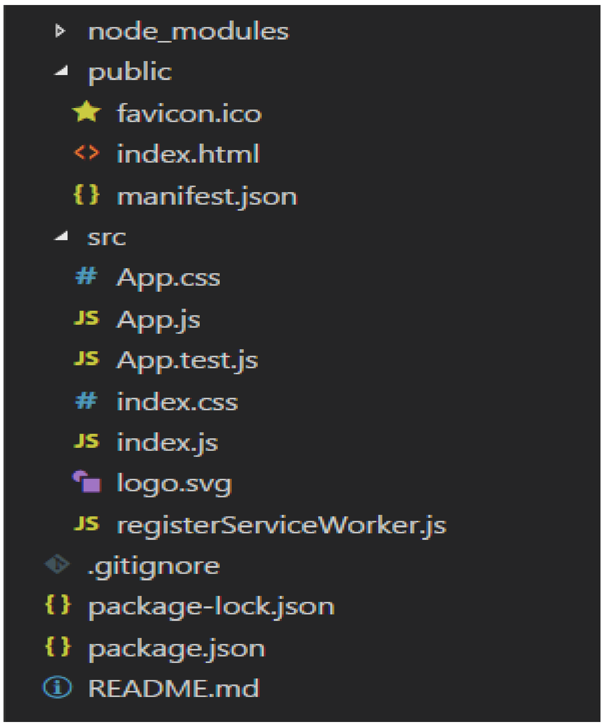

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\jason>node -v
v12.17.0

C:\Users\jason>npm -v
6.14.4

C:\Users\jason>_
```

**Step 2**:

```
npx create-react-app myapp
cd myapp
npm start
```

**Anatomy of React app (Folder structure)**

- README.md is a markdown file that includes a lot of helpful tips and links that can help you while learning to use Create React App.
- node_modules is a folder that includes all of the dependency-related code that Create React App has installed.
- package.json that manages our app dependencies
- .gitignore is a file that is used to exclude files and folders from being tracked by Git.
- public is a folder that we can use to store our static assets, such as images, svgs, and fonts for our React app.
- src is a folder that contains our source code.



```
▸ node_modules
◢ public
    favicon.ico
    index.html
    manifest.json
◢ src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    registerServiceWorker.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

**What is Node.js?**

- Node.js is an open source server environment (free)
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

**What Can Node.js Do?**

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

**What is JSX?**

- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.
- Consider this variable declaration:
- const element = <h1>Hello, world!</h1>;
- JSX is a XML syntax extension. Just like HTML, JSX tags can have a tag names, attributes, and children. If an attribute is wrapped in curly braces, the value is a JavaScript expression.

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

const myFirstElement = <h1>Hello React!</h1>

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

**Expressions in JSX**

With JSX you can write expressions inside curly braces { }

The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result

Ex:

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

const name='niranjan';
const element=<h1>Hello, { name }</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(element);
```

Ex2:

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

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


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

Ex3:

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

const myElement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

**Specifying Attributes with JSX**

JSX allows us to use attributes with the HTML elements.

JSX uses camelcase convention for attributes.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
const myElement = <h1 className="myclass">Hello World</h1>;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

**Specifying Children with JSX**

JSX tags may contain children:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

**Rendering**

React renders HTML to the web page by using a function called render().

The purpose of the function is to display the specified HTML code inside the specified HTML element.

**Rendering Elements**

Rendering an Element in React: In order to render any element into the Browser DOM, we need to have a container or root DOM element. It is almost a convention to have a div element with the id=”root” or id=”app” to be used as the root DOM element. Let’s suppose our index.html file has the following statement inside it.

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

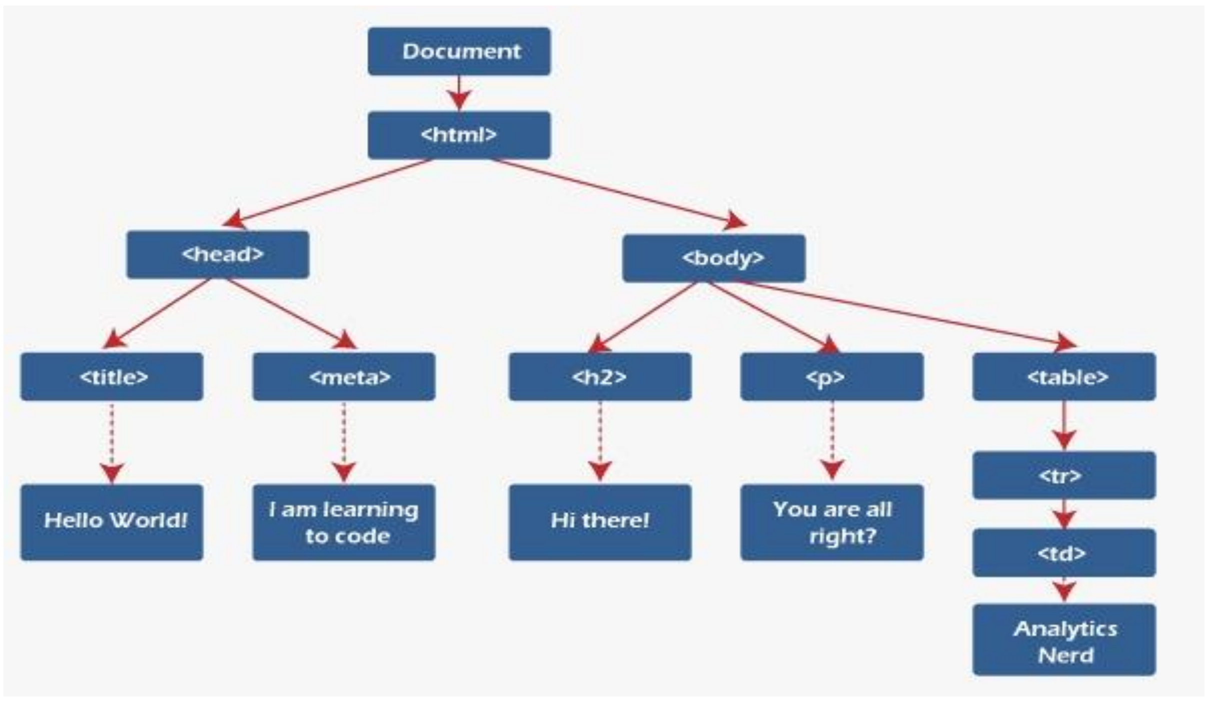
**Filename App.js**: Now, in order to render a simple React Element to the root node, we must write the following in the App.js file.

```
import React,{ Component } from 'react';
class App extends Component {
render() {
        return (
        <div>
                <h1>Welcome to GeeksforGeeks!</h1>
        </div>
        );
}
}       export default App;
```

**What is the DOM?**

The DOM (Document Object Model) represents the web page as a tree structure. Any piece of HTML that we write is added as a node, to this tree.

With JavaScript, we can access any of these nodes (HTML elements) and update their styles, attributes, and so on.



**React-DOM**

ReactDOM is a package that provides DOM specific methods that can be used at the top level of a web app to enable an efficient way of managing DOM elements of the web page.

**Pre-requisite:** To use the ReactDOM in any React web app we must first import ReactDOM from the react-dom package by using the following code snippet:

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

**render() Function**

This function is used to render a single React Component or several Components wrapped together in a Component or a div element.

```
ReactDOM.render(element, container, callback)
```

**Parameters**: This method can take a maximum of three parameters as described below.

- **element:** This parameter expects a JSX expression or a React Element to be rendered.
- **container:** This parameter expects the container in which the element has to be rendered.
- **callback:** This is an optional parameter that expects a function that is to be executed once the render is complete.

**Virtual DOM in React**

Virtual DOM in React is a “virtual” representation of the actual DOM. It is nothing but an object created to replicate the actual DOM.

**What are React Components?**

- React components are independent and reusable code.
- They are the building blocks of any React application.
- Components serve the same purpose as JavaScript functions, but work individually to return JSX code as elements for our UI.
- Components usually come in two types, functional components and class components

**Functional Components**

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

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

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

**Class Components**

A class component must include the extends React.Component statement.

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

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

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

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

**Rendering a Component**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

**Composing Components**

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div> );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

**Create your first React Component.**

When creating a React component, the component's name *MUST* start with an upper case letter.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

**What are props?**

Props is short for properties and they are used to pass data between React components. React's data flow between components is uni-directional (from parent to child only).

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

class ParentComponent extends React.Component {
  render() {
    return (
      <ChildComponent name="First Child" />
  );
  }
}
const ChildComponent = (props) => {
  return <p>{props.name}</p>;
};
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<ParentComponent />);
```

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

function Name(props) {
  return <h2>My name { props.name }!</h2>;
}

function Display() {
  return (
    <>
      <Name name="Niranjan" />
    </>
  );
}

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

**What is state?**

React has another special built-in object called state, which allows components to create and manage their own data. So unlike props, components cannot pass data with state, but they can create and manage it internally.

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

class Test extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            id: 1,
            name: "test"
        };
    }

    render() {
        return (
            <div>
              <p>{this.state.id}</p>
              <p>{this.state.name}</p>
            </div>
        );
    }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Test />);
```

**Communication between components using Props**

You can pass props down to child components.

```
import React from 'react';
class Parent extends React.Component {
  render() {
    const variable = 5;
    return (
      <div>
        <Child message="message for child" />
        <Child message={variable} />
      </div>
    );
  }
}
```

```
class Child extends React.Component {
   render() {
      return <h1>{this.props.message}</h1>
   }
}
export default Parent;
```

**What is the React component lifecycle?**

In React, components go through a lifecycle of events:

1. Mounting (adding nodes to the DOM)
2. Updating (altering existing nodes in the DOM)
3. Unmounting (removing nodes from the DOM)
4. Error handling (verifying that your code works and is bug-free)

**Mounting**

Mounting means putting elements into the DOM.

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

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

The render() method is required and will always be called, the others are optional and will be called if you define them.

**constructor**

The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component).

**getDerivedStateFromProps**

The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM.This is the natural place to set the state object based on the initial props.

It takes state as an argument, and returns an object with changes to the state.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
 constructor(props) {
   super(props);
   this.state = {favoritecolor: "red"};
 }
 static getDerivedStateFromProps(props, state) {
   return {favoritecolor: props.favcol };
 }
 render() {
   return (
     <h1>My Favorite Color is {this.state.favoritecolor}</h1>
   );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header favcol="yellow"/>);
```

**render**

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

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

**componentDidMount**

The componentDidMount() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

**Updating**

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**Unmounting**

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

- componentWillUnmount()

**Error Handling**

**Error Handling** methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

- static getDerivedStateFromError()
- componentDidCatch()

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

import { Component } from "react";

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

```
export default ErrorBoundary;
```

## Handling Events

React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={shoot}  instead of onClick="shoot()".

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

function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }
  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

## Passing Arguments

To pass an argument to an event handler, use an arrow function.

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

function Football() {
  const shoot = (a) => {
    alert(a);
  }
  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

**Conditional Rendering**

if Statement

We can use the if JavaScript operator to decide which component to render.

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

function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}
function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
// Try changing to isLoggedIn={true}:
root.render(<Greeting isLoggedIn={false} />);
```

**Ternary Operator**

condition ? true : false

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

function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}
function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  return (
    <>
      { isLoggedIn ? <UserGreeting/> : <GuestGreeting/> }
    </>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
// Try changing to isLoggedIn={true}:
root.render(<Greeting isLoggedIn={true} />);
```

**Lists & Keys**

**Rendering Multiple Components**

You can build collections of elements and include them in JSX using curly braces {}.

- The list is used to display data in an ordered format and is traversed using the map() function.
- lists are a continuous group of text or images.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =><li>{number}</li>);
  return (
    <ul>{listItems}</ul>
  );
}
const numbers = [1, 2, 3, 4, 5];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<NumberList numbers={numbers} />);
```

**Keys**

- Keys are used to identify which items have changed, are added, or are removed.
- Keys should be given inside the array to give the elements a stable identity.

```
import React from 'react';
import ReactDOM from 'react-dom';
function App(props) {
const numbers = props.numbers;
return (
<div>
<ul>
   {numbers.map((number) =>
    <li key={number.toString()}>{number}</li>
    )}
</ul>
</div>
);
}
const numbers = ['Indore', 'Mumbai', 'Pune', 'Hyderabad', 'Banglore'];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App numbers={numbers} />);
export default App;
```

**Use of HTML tags in forms like select,input, file, textarea, etc.**

```
<html>
<head>
<title>Form Elements</title>
</head>
<body>
<form>
<lable>Text Box</lable>
<input type="text" id="t1" name="name" value=""/><br><br>
Radio Button: <br>
<input type="radio" id="r1" name="" value=""/>Male<br> <br>
<input type="radio" id="r1" name="" value=""/>FeMale<br><br>
Check Box:<input type="checkbox" id="c1" name="" value=""/><br><br>
File:<input type="file" id="e1" name="file" value=""/><br><br>

Select:<br>
<label>Sem</label>
<select name="sem" id="sem">
  <option value="1">1 Sem</option>
  <option value="2">2 Sem</option>
</select><br><br>

Text Area:<br>
<textarea id="ta1" name="textarea" rows="4" cols="50">
At w3schools.com you will learn how to make a website.
</textarea><br><br>

<fieldset>
    <legend>Personal Details:</legend>
    <label>First name:</label>
    <input type="text" id="fname" name="fname"><br><br>
    <label>Last name:</label>
    <input type="text" id="lname" name="lname"><br><br>

  </fieldset><br><br>

Button:<input type="button" id="t1" name="" value="Submit"/><br>
</form>
</body>
</html>
```

**Controlled components**

- 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 input Tag**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
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();
```

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

**The textarea Tag**

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

**The select Tag**

```
render() {
    return (
      <form onSubmit={this.handleSubmit}>
```

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

**The file input Tag**

```
<input type="file" />
```

**Uncontrolled components**

- In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.
- To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

**The input Tag**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }
  handleSubmit(event) {
    alert('A name was submitted: ' + this.input.current.value);
    event.preventDefault();
```

```
 }
 render() {
   return (
     <form onSubmit={this.handleSubmit}>
       <label>
         Name:
         <input type="text" ref={this.input} />
       </label>
       <input type="submit" value="Submit" />
     </form>
   );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<NameForm />);
```

**The file input Tag**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class FileInput extends React.Component {
 constructor(props) {
   super(props);
   this.handleSubmit = this.handleSubmit.bind(this);
   this.fileInput = React.createRef();
 }
 handleSubmit(event) {
   event.preventDefault();
   alert(
     `Selected file - ${this.fileInput.current.files[0].name}`
   );
 }
 render() {
   return (
     <form onSubmit={this.handleSubmit}>
       <label>Upload file:
         <input type="file" ref={this.fileInput} />
       </label>
       <br />
       <button type="submit">Submit</button>
     </form>
   ); }}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FileInput />);
```

**Lifting State Up**

In React, it is recommended to make a common pattern while using the same state in multiple components. “Lifting state up” is a common pattern used in React to share state among multiple components to avoid complex and unnecessary patterns for state management.

**Testing single page application**

Filename: App.css

```
.App {
  text-align: center;
  background-color:green;
  }
  .label{
  display: block;
  font-size: larger;
  color: white;
  padding: 5px;
  }
  .input{
  font-size: larger;
  padding: 5px;
  margin: 2px;
  }
  .btn{
  color: white;
```

```
 display: block;
 margin: 10px auto;
 }
 .messages{
 display: flex;
 justify-content: center;
 }
```

Filename: App.js

```
import './App.css';
import Form from "./Form"
function App() {
return (
   <div className="App">
   <Form />
   </div>
);
}
export default App;
```

**Filename: Form.js**

```
import { useState } from 'react';
export default function Form() {
// States for registration
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
// States for checking the errors
const [submitted, setSubmitted] = useState(false);
const [error, setError] = useState(false);

// Handling the name change
const handleName = (e) => {
        setName(e.target.value);
        setSubmitted(false);
};
// Handling the email change
const handleEmail = (e) => {
        setEmail(e.target.value);
        setSubmitted(false);
};
// Handling the password change
const handlePassword = (e) => {
        setPassword(e.target.value);
        setSubmitted(false);
};
// Handling the form submission
const handleSubmit = (e) => {
        e.preventDefault();
        if (name === '' || email === '' || password === '') {
        setError(true);
        } else {
        setSubmitted(true);
        setError(false);
        }
};
// Showing success message
const successMessage = () => {
        return (
        <div
                className="success"
                style={{
                display: submitted ? '' : 'none',
                }}>
                <h1>User {name} successfully registered!!</h1>
        </div>
        );
};

// Showing error message if error is true
const errorMessage = () => {
        return (
        <div
                className="error"
                style={{
                display: error ? '' : 'none',
                }}>
                <h1>Please enter all the fields</h1>
        </div>
        );
};
return (
        <div className="form">
        <div>
                <h1>User Registration</h1>
        </div>
        {/* Calling to the methods */}
        <div className="messages">
                {errorMessage()}
                {successMessage()}
        </div>
        <form>
                {/* Labels and inputs for form data */}
                <label className="label">Name</label>
                <input onChange={handleName} className="input"
                value={name} type="text" />
                <label className="label">Email</label>
                <input onChange={handleEmail} className="input"
                value={email} type="email" />
```

```
                    <label className="label">Password</label>
                    <input onChange={handlePassword} className="input"
                    value={password} type="password" />
                    <button onClick={handleSubmit} className="btn" type="submit">
                    Submit
                    </button>
          </form>
          </div>
);
}
```