**Installation:**

| Node js - https://nodejs.org/en/download/ |
| --- |

NPM – Once node.js is installed, it will be there by default

| npm install -g npx |
| --- |

**To Create a React Application:**

| npx create-react-app appName |
| --- |

appName - Name of the application that you are going to create

**To Run React Application:**

npm start

**Component:**

Things to remember - Component & Views

We’ll user - Functions (return html) & Classes

ReactDom.render(<**whatToRender**>, <**whereToRender**>);   
  
React.createElement(<**htmlElement**>, <**propertyforElement**>, <**contentInsideElement**>)

Adjacent element inside return should be wrapped inside a single element

| return(   <**div**>  <**h1**>Hello</**h1**>  </**div**> ) |
| --- |

In react, everything is javascript -> JSX (it's not an html)

**Component Type:**

Functional Component - It doesn’t know what is happening with application, it don’t have state ( stateless )

Class base Component - It knows what is happening with in the application, by default it have state ( stagefull )

\*\* props is available for both Functional & Class base component

Syntax:

| class <className> extends React.Component {    render( ) {    return ( ... )    }   } |
| --- |

{ Component } - ES6 => it will get the value of name Component and Assign it to a Variable -> Component

const Component = React.Component (creating alias)

**Styling**

Inline Styling - create a javascript object and define our style (camelCase)

External Styling

**Event Methods**

An event represents the precise moment when something happens.

Types of Event

1. Mouse Events :

onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit, onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave

onMouseMove onMouseOut onMouseOver onMouseUp

2. Keyboard Events:

| onKeyDown onKeyPress onKeyUp |
| --- |

3. Form Events:

4. Document/Window Events

| Mouse Events - click, dblclick, mouseenter, mouseleave   Keyboard Events - keypress, keydown, keyup   Form Events - submit, change, focus, blur   Document/Window Events - load, resize, scroll, unload      onClick={ this.functionName }   onClick={ ( ) => this.functionName( ) } |
| --- |

Function Context

//Function

*function* onLoad(){

*var* student = {}

console.log(this) //Global Object

}

onLoad()

*var* onUnLoad = *function*(){

console.log(this) //Global Object

}

onUnLoad()

//Method

*var* video = {

name: 'big-bunny.mp4',

size: '4MB',

onPlay(){

console.log(this)

}

}

video.onPlay() // this will refer the video object

//Constructor Function

*function* OnPlayAudio(){

console.log(this)

}

new OnPlayAudio() // this will refer the OnPlayAudio object

constructor - super( ) => when the parent class has a constructor and we are extending that to the child. If we use constructor

in child we need to define super( ) - which will include the parent constructor to its child class

**State**

It's a object, which act as a database

It looks like a normal object, but when we change its value, React will re-render the component

state = { key : value } //declaration - at the constructor => this.state = { }   
  
this.setState( { key : value } ) //To update the state

Try to reduce using it, coz it will required lot of memory

Bind(this)

When we used to trigger a function and we need to pass the global reference, so we use <functionName>.bind( this )

\*\* To avoid explicit bind( this ) - we need to use ES6 Flat head arrow function. It will always point this - as global reference

**React Routing:**

Npm : https://www.npmjs.com/package/react-router-dom   
  
Installation: npm install react-router-dom --save

Route Configuration:

import { BrowserRouter, Route, Switch } from 'react-router-dom';

| <**BrowserRouter**>  <**Switch**>  <**Route** path="pathName" component={componentName} /> </**Switch**> </**BrowserRouter**> |
| --- |

Navigation by code:

| this.props.history.push('path', parameter) |
| --- |

Navigation by HTML:

| import { NavLink } from 'react-router-dom' |
| --- |

| <NavLink  to="path"  activeClassName=""  className=""></NavLink> |
| --- |

Get Value from URL:

| this.props.match.params |
| --- |

Get Value from Route Object:

| this.props.history.location.state |
| --- |

Lazy Loading a Component:

| Methods - lazy, Suspense |
| --- |

Syntax:

| const HeaderComponent = lazy(() => import('../components/header')); |
| --- |

| <**Route** path="/home" render={ () => {  return (  <**Suspense** fallback={<h2>Loading...</**h2**>}>  <**HeaderComponent**></**HeaderComponent**>  </**Suspense**>  )  } }/> |
| --- |

**Run React app in different Port:**

Update the port number in package.json at the start value,

| "start": "set PORT=3006 && react-scripts start" |
| --- |

**React Lifecycle :**

Each component in React has a lifecycle which you can monitor and manipulate during its four main phases. The four phases are: Initialization, Mounting, Updating, and Unmounting.

- **4 phase**

1. Initialization

a) constructor() //set default value for state

2. Mounting

| a) componentWillMount() //calls before render    b) render() //render method    c) componentDidMount() //calls after render |
| --- |

3. Updating

| a) shouldComponentUpdate(nextProps, nextState)  //calls whenever state changed    b) componentWillReceiveProps(nextProps)  // calls whenever props received    c) componentWillUpdate() // calls before any update    d) render() // render method   e) componentDidUpdate() // calls after any update |
| --- |

4. Unmounting

a) componentWillUnmount() // calls when component get destroyed

**Getting Information from Server**

1. Client makes a http (HyperText Transfer Protocol) request through

browser and server sends json/xml formatted data.

2. For making http request

- XmlHttpRequest

- Axios

- Windows fetch / Fetch

**API**:

Application Programming Interface (API) is an interface or communication protocol between different parts of a computer program

**REST API**:

REpresentational State transfer which essentially refers to a style

of web architecture

**REST Methods**:

GET - To get Information from the server

POST - To send Information to the server (Create)

PUT - To send Information to the server (Update / Edit)

DELETE - To delete Information from the server

**Installation**:

| npm install axios --save |
| --- |

Usage:

import axios from 'axios';

Syntax:

| axios.<method>(url, parameter)    .then((response) => {    //success code    })   .catch((error) => {    //error code   }) |
| --- |

**Conditional Rendering:**

- conditional based view rendering based on the state of application.

Types:

Element Variables

| - { variable } |
| --- |

Inline If with Logical && Operator

| - { condition && JSX} |
| --- |

Inline If-Else with Conditional Operator

- { condition ? JSX 1 : JSX 2 }

**Pure Component:**

- It automatically checks whether a re-render is required for the component.

- It changes the life-cycle method shouldComponentUpdate

- increase in performance because it reduces the number of render operation in the application

**syntax**:

| import { PureComponent } from 'react';   //code   class Home extends PureComponent {   } |
| --- |

**Type Checking With PropTypes**

- PropTypes exports a range of validators that

can be used to make sure the data you receive is valid.

Installation:

- npm i prop-types --save

List of important PropTypes:

PropTypes.string,

PropTypes.string.isRequired,

PropTypes.oneOfType([

PropTypes.string,

PropTypes.number

]),

PropTypes.string,

PropTypes.number,

PropTypes.bool,

PropTypes.object,

PropTypes.shape({

PropTypes.number,

area: PropTypes.string,

city: PropTypes.string

}),

PropTypes.array

PropTypes.arrayOf(PropTypes.string),

PropTypes.func

**Higher-Order Components:**

- A higher-order function is a function that can take another function as an argument and/or that returns a function as a result.

- Higher-order component is a function that takes (wraps) a component and

returns a new component.

- HOCs are not part of the React API. (pattern)

**Refs and the DOM (Uncontrolled)**

Refs provide a way to access DOM nodes or React elements created in the render method.

When to Use Refs

There are a few good use cases for refs:

Managing focus, text selection, or media playback.

Triggering imperative animations.

Integrating with third-party DOM libraries.

Avoid using refs for anything that can be done declaratively.

**Passing Data Between React Components**

Parent

Children

Siblings / Unrelated

1. From Parent to Child using Props

2. From Child to Parent using Callbacks

3. React Router

4. Between Siblings / Unrelated:

(i)Combining the Parent to Child and Child to Parent method

* It will not work for complicated directory structures.

(ii) Using React Context API

* The React Context API has been around as an experimental feature, available after 16.3.0
* Prop-drilling refers to the technique of passing down variables to sub components.
* The main idea is functional programming where you pass the parameters to the next function and so on

(iii) Using Redux

- Use a global store maintaining the states of all child components which are needed to interact and consume the data required from the store

Material Design

It is inspired by the physical world and its textures, including how they reflect light and cast shadows. Material surfaces reimagine the mediums of paper and ink.

**React Material UI**

React components for faster and easier web development. Build your own design system, or start with Material Design.

| Url: https://material-ui.com/   Installation:   npm install @material-ui/core -- save |
| --- |

Font

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" />

Icons

<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />

**Redux Framework**

Redux - https://redux.js.org/

React Redux - https://react-redux.js.org/

To setup redux we need to install the following plugins,

1. npm install redux --save

2. npm install react-redux --save

**Unit Testing**

React Test Renderer:

It used to render React components to pure JavaScript objects, without depending on the DOM, also it makes it easy to grab a snapshot of the "DOM tree"

Enzyme:

It is a JavaScript Testing utility for React that makes it easier to test your React Components' output.

Installation:

1. npm i enzyme --save-dev
2. npm i jest-enzyme --save-dev
3. npm i enzyme enzyme-adapter-react-16 --save-dev
4. npm i react-test-renderer --save-dev

Matchers

Truthiness:

toBeNull - matches only null

toBeUndefined - matches only undefined

toBeDefined - is the opposite of toBeUndefined

toBeTruthy - matches anything that an if statement treats as true

toBeFalsy - matches anything that an if statement treats as false

Equality

toBe

toEqual

Negotiation

not

Numbers

toBeGreaterThan

toBeGreaterThanOrEqual

toBeLessThan

toBeLessThanOrEqual

Strings

toMatch

stringContaining(string)

Object

objectContaining(object)

Arrays

toContain

Mock

toHaveBeenCalled - To ensure that a mock function got called.

toHaveBeenCalledTimes(number) - To ensure that a mock function got called an exact number of times.

toHaveBeenCalledWith(arg) - To ensure that a mock function was called with specific arguments

toHaveReturned - To test that the mock function successfully returned

toHaveReturnedWith - To ensure that a mock function returned a specific value.

Exceptions

toThrow

Jest Mocks

There are 3 main types of mocking in Jest

1. jest.fn : Mock a function

2. jest.mock : Mock a Module

3. jest.spyOn : Spy or Mock a function

Component:

import Adapter from 'enzyme-adapter-react-16';

import { shallow, configure } from 'enzyme';

configure({adapter: new Adapter()});

const component = shallow(<componentName />);

expect(component.instance().focus()).toBe(true);