React/ReactJS
==========
It is a declarative, efficient and flexible javascript frontend library responsible for building frontend applications or User interfaces(UI).
It is an open source, component based frontend library responsible only for view layer of the application.
It was developed by Jordan Walke who was a software engineer at Facebook.
It was initially developed and maintained by Facebook and later it is used in their own products like Whatsapp and Instagram.
React was released to the public in the month of May, 2013.
The latest version of React/Reactjs is v18.2.2.
The official website of React is http://www.reactjs.org.
React is used to create a reusable component.
A component is a building block of react application.

Advantages of React/ReactJS
==========

1) It is easy to learn and easy to use.
2) It supports one way data binding.
3) It supports Virtual DOM.
4) It supported by all major browsers.
5) It creates reusable components.
6) Good Documentation and Community support.

Q)Differences between Angular and React?

Angular	React
It was released in October,2010.	It was released in May, 2013.
Angular is a product of Google.	React is a product of Facebook.
Angular is an open source javascript framework for web and mobile development.	React is a open source frontend javascript library responsible only for view layer of the application.
Typescript language is used in angular.	JSX language is used in react.
Jasmine and Karma used as a testing frameworks.	Jest and Enzyme used as a testing frameworks.
It supports two way data binding.	It supports one way data binding.
It supports Traditional DOM.	It supports Virtual DOM.
The default port number is 4200.	The default port number is 3000.
It is used for large scale and rich featured applications.	It is used for SPA (Single Page Application).

Angular used by Google, Nike, McDonoalds, paypal,Gmail and etc.

React used by Facebook,whatsapp, instagram,airbnb and etc.

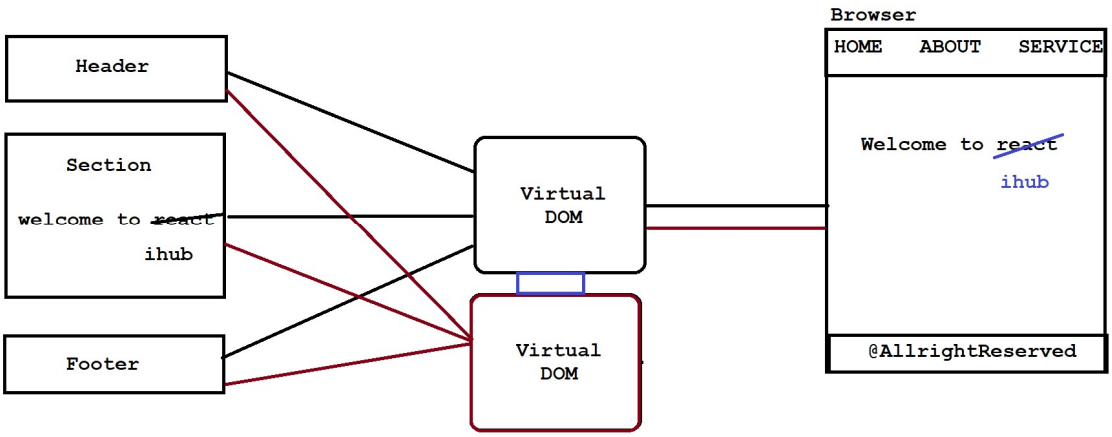
How ReactJS works internally

==========

React uses a virtual DOM that is basically a DOM tree representation in JavaScript.

So when it needs to read or write to the DOM, it will use the virtual representation of it.Then the virtual DOM will try to find the most efficient way to update the browser's DOM.

Assume we have created multiple components and consistently we are performing some changes in our application.Now we need to see ,how virtual DOM react on each change.



Pre-requiste to learn React/ReactJS

==========

1) Basics of HTML,CSS,Javascript and Bootstrap.
2) Strong knowledge on JSX.
3) Usage of npm commands.
4) Basics on ES6 standards.

What is JSX

==========

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX tags having a tagname, attributes and children.

JSX is not a necessity to write React applications.Instead we can use Babel.

JSX makes your react code simpler and elegant.

JSX ultimately transpiles to pure JavaScript which is understood by the browsers.

JSX Elements

==========

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.

ex:1

JSX code

```
<h1>IHUB Talent</h1>
Here h1 is a jsx element.
```

Babel code

```
React.createElement("h1",null,"IHUB Talent");
```

Here h1 is a tag name.
Here null is a optional property name.
Here IHUB Talent is a text.

ex:2

JSX code

```
<div>
        <h1>Hello React JS </h1>
</div>
```
babel code

```
React.createElement("div",null,
                React.creteElement("h1",null,"Hello ReactJs"));
```
ex:3

JSX code

```
<div id="myId">
        <h1>Hello React JS </h1>
</div>
```
babel code

```
React.createElement("div",{id:'myId'},
                React.creteElement("h1",null,"Hello ReactJs"));
```
ex:4

JSX code

```
<div class="myClass">
        <h1>Hello React JS </h1>
</div>
```
babel code

```
React.createElement("div",{class:'myClass'},
                React.creteElement("h1",null,"Hello ReactJs"));
```
Note:

In above code , warning message will be displayed on console i.e
Invalid DOM property 'class'.
In order to remove this warning from our application we need to use "className" attribute.
In javascript, "class" is a keyword which is used to create React components.
In react , CSS class name must specify by "className" attribute.

ex:5

JSX code

```
    <div id="myId" class="myClass">
            <h1>Hello React JS </h1>
    </div>
```
babel code

```
    React.createElement("div",{id:'myId',className:'myClass'},
                    React.creteElement("h1",null,"Hello ReactJs"));
```
JSX Expressions
======================
JSX allows us to 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:1

```
    let name="Alan Morries";
    <h1>My Name is {name}</h1>
```
ex:2

```
    <h1>The value is = {5+5} </h1>
```
ex:3

```
    <h1>{Math.random()*100}</h1>
```
ex:4

```
    <h1>{Math.floor(Math.random()*100)}</h1>
```
Note:
------
we can't use conditional statements like if, while and etc inside JSX expression.

NPM
=====
NPM stands for Node Package Manager.
It is integrated tool for NodeJs.
It is used to download node modules/dependencies/libraries.
We can download any mododule as follow.
ex:
```
    cmd> npm  install  -g  node_module/dependency/library
```

All the modules will downloaded inside "node_modules" folder.

Setup for npm command
----
1) Download and Install NodeJS module.
  https://nodejs.org/en/download

2) Copy the nodejs directory from "C/program files" drive.
  ex:
    C:\Program Files\nodejs

3) Paste nodejs directory in environment variables.
  ex:
    right click to my computer --> properties --> advanced system settings -->
    environmental variables --> user variables --> click to new button -->
    variable name : path
    variable value : C:\Program Files\nodejs; -->ok -->ok -->ok.

4) Open the command prompt and check below commands.
  ex:
    cmd> npm -version
    cmd> node --version

First application development using React/ReactJS
==========
step1:

    Make sure Nodejs setup has done perfectly.
step2:

    Download and Install VSC(Visual Studio Code) editor.
    ex:
            https://code.visualstudio.com/
step3:

    Create a "Reactprojects" folder inside "E" drive.
step4:

    Open the command prompt from "Reactprojects" folder.
step5:

    Open the visual studio code editor from "Reactprojects" folder.
    ex:
            Reactprojects> code .
step6:

    Install "create-react-app" module for creating react applications.
    ex:
            Reactprojects> npm install -g create-react-app
step7:

    Create a "myapp1" react project in VSC editor.
    ex:
            Reactprojects> create-react-app myapp1

step8:

Switch to myapp1 project.
ex:
Reactprojects>cd myapp1
step9:

Run the myapp1 project.
ex:
Reactprojects/myapp1> npm start
step10:

Test the react application/project.
ex:
http://localhost:3000
Note

By default react application runs on a light weight development server with 3000 port number.

Explaination of React project and Work flow
==========
myapp1
|
|----node_modules
|
|----public
| |
| |---favicon.ico
| |---index.html
| |---manifest.json
| |
|
|-----src
| |
| |---App.css
| |---App.js
| |---App.test.js
| |---index.css
| |---index.js
| |---logo.svg
| |
|---package.json
|---README.md

"myapp1" is a Name of a project.
"node_modules" contains all packages and dependencies installed.
"favicon.ico" is a favourite icon for a web site.
"index.html" file holds HTML template of our application(Main template).

"manifest.json" file provides metadata used when your web app is
installed on a user's mobile device or desktop.
"App.css" is a css file related to App.js but it's global.
"App.js" is parent component of our React app.
"App.test.js" is for test environment.
"index.css" is a css file related to index.js but it's global.
"index.js" is a javascript entry point.
"logo.svg" is a React logo.
"package.json" contains all dependencies used in React app along with their versions..
Note:

    To build the project, "index.js" and "index.html" must exist
    with exact file name(mandatory).

    Above two files are mandatory at the time of deployment
    not at the time of development.

            load to   render to   output
    App.js--------->index.js---------->index.html------->browser

Second Application development using React/ReactJS
==========
myapp2
|
|---node_modules
|
|
|-----public
    |
    |---favicon.ico
    |---index.html (3)
    |---manifest.json
|
|-----src
    |
    |---index.js (2)
    |
    |---App.js (1)
|
|-----package.json
|-----README.md
step1:

    Create a myapp2 react application/project.
    ex:
            Reactprojects> create-react-app myapp2
            or
            Reactprojects> npx create-react-app myapp2

step2:

Open the VSC editor from Reactprojects folder.
ex:
Reactprojects> code .
step3:

Delete App.css,App.js,App.test.js,index.js and index.css file
from src folder.
step4:

Create index.js file inside "src" folder.
ex:
index.js

```
import App from "./App";
import React from "react";
import ReactDOM from "react-dom/client";
const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
        <React.StrictMode>
        <App/>
        </React.StrictMode>
        )
```
step5:

Create App.js file inside "src" folder.
ex:
App.js

```
function App()
{
        return(

            <h1>I Love ReactJS </h1>
        )
}
export default App;
```
step6:

Switch to myapp2 project.
ex:
Reactprojects> cd myapp2
step7:

Run the myapp2 project.
ex:
Reactprojects/myapp2> npm start

step8:

Test the application by using below request url.
ex:
http://localhost:3000

React Fragment
==========
Fragment is used to group of list of childrens without adding
extra nodes of the DOM.
In general, We can return
only one element at a time but we can't return more then one element directly.
To return more then one element we need to use React Fragment.
syntax

<React.Fragment>
-
-
</React.Fragment
or
<>
-
-
</>
Examples

App.js

```
function App
{
  return (
    //return react element
    return  <h1>IHUB Talent</h1>
          <h2>React Tutorial For Freshers</h2>
  );
}
//export React component
export default App
```

o/p: Filed to compile

To overcome above problem we can use tag and inside that
 tag we can declare any child tags.
ex:
App.js

```
function App
{
```

```
   return (
      //return react element
      return
               <div>
                        <h1>IHUB Talent</h1>
               <h2>React Tutorial For Freshers</h2>
               </div>
   );
}
//export React component
export default App
```

Note:

In above program " " tag is a unused tag.

To remove unused/unnecessary tags we can use React Fragment.

approach1

App.js

```
import React from "react";
function App()
{
   return (
         <React.Fragment>
         <h1>IHUB React Tutorial</h1>
         <h1>React Classes for Freshers</h1>
         </React.Fragment>
   );
}
export default App;
```

approach2

App.js

```
import React from "react";
import {Fragment} from 'react';
function App()
{
   return (
         <Fragment>
         <h1>IHUB React Tutorial</h1>
         <h1>React Classes for Freshers</h1>
         </Fragment>
   );
}
export default App;
```

approach3

App.js

```
import React from "react";
function App()
{
   return (
        <>
        <h1>IHUB React Tutorial</h1>
        <h1>React Classes for Freshers</h1>
        </>
   );
}
export default App;
```

React Components
==========
Components are Building blocks of any React app.
Component allows us to split UI into independent reusable pieces.
ex:
    navbar, header, footer , body and etc.

Components are like Javascript functions.They accept arbitrary inputs called "props" and return React Element describing what should appears on the screen.
A Component name always starts with capital letter.
ex:
      represent as HTML div tag.
     But <Div> represent a component in react.

we can create react component in two ways.

1)Function Component /functional component
2)Class Component

1)Function Component
==========
It is a Javascript function which accept single "props" object as argument
with data and returns a React Element.
The functional component is also known as a stateless component because they do not hold or manage state.
syntax

```
function function_name()
{
    return element;
}
```

Project Directory structure

```
myapp3
|
|----node_modules
|
|----public
|    |
|    |---favicon.ico
|    |---index.html (3)
|    |---manifest.json
|    |
|
|-----src
|    |
|    |---index.js   (2)
|    |---App.js (1)
|    |
|---package.json
|---README.md
```

step1:

Develop React Application

ex:

E:/BUI-2pm/ReactProjects>create-react-app myapp3

step2:

Delete all the starting 6 files from src folder.

step3:

---

create a App.js file in src folder to develop a function component.

Student.js

```
function App()
{
  return <h1>Function Component Example</h1>
}
export default App;
```

Note:

Above code is applicable for older versions and for lastest versions like EC6 ,we use below code.

ex:

App.js

```
const App=()=>{
   return <h1>Function Component Example</h1>
}
export default App;
```

step4:
----
create "index.js" file to render the component to index.html file.
index.js

```
import App from './App';
import ReactDOM from 'react-dom/client';
import React from 'react';
const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrickMode>
    <App/>
    </React.StrickMode>
)
```
step5:
----
    move to myapp3
    ex:
            cmd/ReactProjects>cd myapp3
step6:

    Run the Application
    ex:
            cmd/myapp3> npm start
step7:

    Test the Application.
    ex:
            http://localhost:3000

Function component with props
==========
In order to use props in a component We need to perform following changes in react "myapp3" project.
syntax

```
function fun_name(props)
{
    return React Element
}
```
ex:1

App.js

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

```
export default App;
```
or

App.js

```
const Student=(props)=>
{
  return <h1>Hello {props.name}</h1>
}
export default App;
```

Rendering the Component

---

we can render the component in index.js file as given below.

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrickMode>
    <App name="Alan"/>
    </React.StrickMode>
)
```

ex2:

App.js

```
const App=(props)=>
{
  return (
        <>
        <h1>First Name : {props.firstName}</h1>
        <h1>Second Name: {props.lastName}</h1>
        </>
  )
}
export default App;
```

index.js

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

```
const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrickMode>
            <App firstName="Alan" lastName="Morries"/>
    </React.StrickMode>
)
```

2)Class Component

==========

A class Component requires to extends from React Component.
The class must implements a render() method function which returns A react Element to be render.This is Similar to return value of a functional component.
In a class based component props are accessible via this.props.
The class component is also known as a stateful component because they can hold or manage local state.

syntax

```
class Class_name extends Component
{
    render()
    {
            return element.
    }
}
```

Project structure

```
myapp4
|
|------node-modules
|
|------public
|    |
|    |------favicon.ico
|    |------index.html (3)
|    |------manifest.json
|
|------src
     |
     |------index.js(2)
     |------App.js (1)
|    |
|
|------package.json
|------README.md
```

step1:

Develop React Application.
ex:
cmd/ReactProjects>create-react-app myapp4
step2:

Delete all the files from src folder.
step3:
---
create a "App.js" file in "src " folder.
App.js

```
import { Component } from "react";
export default class App extends Component
{
 render()
 {
  return <h1>First Class Component</h1>
 }
}
```
step4:
---
create "index.js" file to render the output to index.html file.

index.js

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

const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>
)
```
step5:
---
move to myapp4
ex:
cmd/ReactProjects> cd myapp4
step6:

Run the application.
ex:
cmd/ReactProjects/myapp4>npm start

step7:

Test the React Application.
ex:
http://localhost:3000

Class component with props
==========
In order to use props in a class component we need to perform following changes.
App.js

```
import { Component } from "react";
export default class App extends Component
{
 render()
 {
  return <h1>Name : {this.props.name}</h1>
 }
}
```
index.js

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

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

Composing Components in React
==========
A component can refer to other components in their output is called composing component.
Let us use some component abstraction for any level of details.

Project structure
---
myapp4
|
|-----node_modules
|
|-----public
| |
   |---index.html (main template)
   |---favicon.ico (favicon)
   |---manifest.json (metadata)

```
|
|-----src
|    |
     |---index.js (entry point)
     |
     |
     |---App.js  (parent component)
     |
     |---Student.js (custom component)

|
|-----package.json
|-----README.md
```

step1:

Create a React Application.

ex:

ReactProjects> create-react-app myapp4

step2:

Start Visual Studio Code (VSC) Editor.

ex:

ReactProjects> code   .

step3:

Delete all the files from "src" folder.

step4:

Create "index.js" file inside "src" folder.

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

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

root.render(
   <React.StrictMode>
      <App/>
   </React.StrictMode>
)
```

step5:

Create App.js file inside "src" folder.

App.js

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

```
function App()
{
   return (
      <Student/>
   )
}
export default App;
```

step6:
    Create Student.js file inside "src" folder.

Student.js

```
function Student()
{
   return (
      <h1>Student Component</h1>
   )
}
export default Student;
```

step7:
    Move to myapp4.
    ex:
            ReactProjects> cd myapp4

step8:
    Run the react application.
    ex:
            ReactProjects/myapp4> npm start

step9:
    Check the output by using below url.
    ex:
            http://localhost:3000

composing components using props
==========

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

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

root.render(
   <React.StrictMode>
```

```
      <App course="React"/>
   </React.StrictMode>
)
```

App.js

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

function App(props)
{
   return (
      <Student crs={props.course}/>
   )
}
export default App;
```

Student.js

```
function Student(props)
{
   return (
      <h1>My Course Name : {props.crs}</h1>
   )
}
export default Student;
```

React CSS
==========
CSS in React is used to style the React App or Component.

There are three ways available to add styling to your React App or Component with CSS.

1) Inline Styling
2) CSS Stylesheet
3) CSS Module

1)Inline CSS
==========
Inline CSS represent by "style" attribute in React application.

The inline styles are specified with a JavaScript object in camelCase version of the style name.

ex:

App.js

```
import Student from "./Student";

function App()
{
  return <>
        <h1 style={{color:"green"}}>React Inline CSS</h1>
        <h1 style={{backgroundColor:"yellow"}}>React Inline CSS</h1>
         </>
}
export default App;
```

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

App.js

```
import Student from "./Student";

function App()
{
  const mystyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Arial"
  };
  return <>
        <h1 style={mystyle}>React Inline CSS</h1>
        <h1 style={{backgroundColor:"yellow"}}>React Inline CSS</h1>
         </>
}
export default App;
```

2) CSS Stylesheet
==========
We can write styling in a separate file for your React application, and save the file with a .css extension.

Later we can import .css file in our required application.

ex:1

App.js

```
import Student from "./Student";
import './App.css';
function App()
{

  return <>
        <h1>React CSS styles</h1>
        <h1>React CSS styles</h1>
         </>
}
export default App;
```

App.css

```
body{
  background-color: yellow;
}
h1
{
  color:blue;
}
```

ex:2

App.js

```
import Student from "./Student";
import './App.css';
function App()
{

  return <>
        <h1 id="myId">React CSS styles</h1>
        <h1 className="myClass">React CSS styles</h1>
         </>
}
export default App;
```

App.css

```
body{
  background-color: yellow;
}
```

```
#myId
{
  color:blue;
}
.myClass
{
  color:red;
}
```

3. CSS Module
==========
CSS Module is another way of adding styles to your 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 you add can never be applied to other components without your permission, and you never need to worry about name conflicts.

We can create CSS Module with the .module.css extension like a myStyles.module.css.

ex:

App.js

```
import Student from "./Student";
import styles from './mystyles.module.css';
function App()
{

  return <>
        <h1  className={styles.mystyle}>React CSS styles</h1>
        <h1 className={styles.parastyle}>React CSS styles</h1>
         </>
}
export default App;
```

mystyles.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;
}
```