React JS

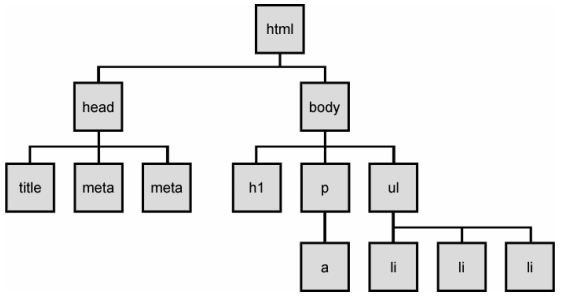
Facebook, a modern web application has over 1.6 billion daily users and handles around 150000 messages and 500000 likes every minute. Updating each like/photo in the UI using JavaScript has significant overhead therefore, data loads slowly and the page becomes unresponsive. Developing and maintaining applications of this size is also difficult.

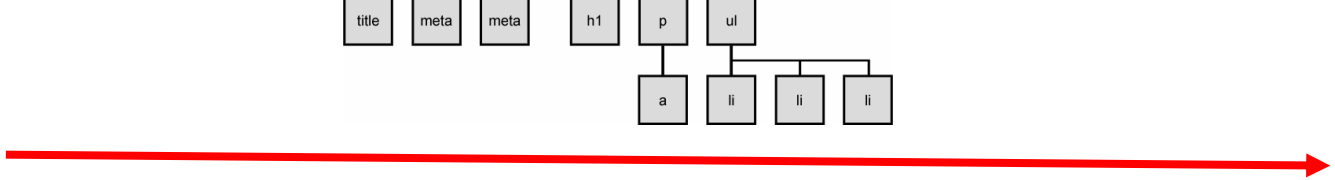
React JS library helps build the user interface of large applications.

Syllabus

How React Works - Page Setup, React Elements, ReactDOM, React, components, React with JSX- React Elements as JSX, Babel, Recipes as JSX, React Fragments, Intro to webpack,
React State Management - Building Forms, **Incorporating Data** - Requesting Data, **Virtualized Lists** - Creating a Fetch Hook, Creating a Fetch Component, Handling Multiple Requests, Memozing Values, React Router.

Javascript





- Java script uses DOM(document Object Model) API to render the content in the browser(to user)
- Java script modifies the DOM tree to display the modified content in the browser.

Creates User Interface(UI) using components and elements using jsx language.

React library

Creates a Virtual DOM using ReactDOM

Virtual DOM compares with Actual DOM and updates/re-renders required elements instead of updating entire tree

Known as Reconciliation

React JS

- React JS is a JavaScript library for creating user interfaces.
- React library provides tools to create elements and components.
- ReactDOM contains tools necessary to render React elements in the browser.
- The DOM API is a collection of objects that JavaScript can use to interact with the browser to modify the DOM.
- React is a library that's designed to update the browser DOM for user specifications.
- The browser DOM is made up of DOM elements. Similarly, the React DOM is made up of React elements

Features

1. Component-based architecture
2. Virtual DOM
3. Unidirectional data flow
4. JSX (java script + XML)
5. SEO performance

React JS

1. React JS is a JavaScript library for creating user interfaces.
2. ReactJS Optimizes the DOM manipulation by writing very simple code.
3. React is used for applications where the data keeps changing very frequently

Features

1. Component-based architecture
2. Virtual DOM
3. Unidirectional data flow
4. JSX (java script + XML)
5. SEO performance

The **create-react-app** tool can be used as it provides a modern build setup allowing to create and run React applications with minimal configuration. The create-react-app is a command-line interface (CLI).

Install create-react-app by running the following command:

```
>npm install -g create-react-app
```

Once the installation is done, create a React app using the below command:

```
>create-react-app my-app
```

Creation of React Application using tools

1. Using create-react-app https://create-react-app.dev/docs/getting-started/

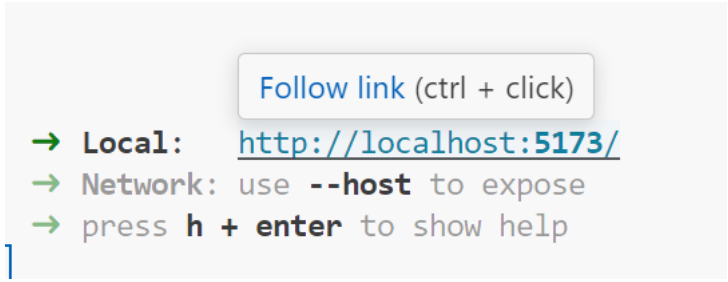
```
npx create-react-app my-app
cd my-app
npm start
```

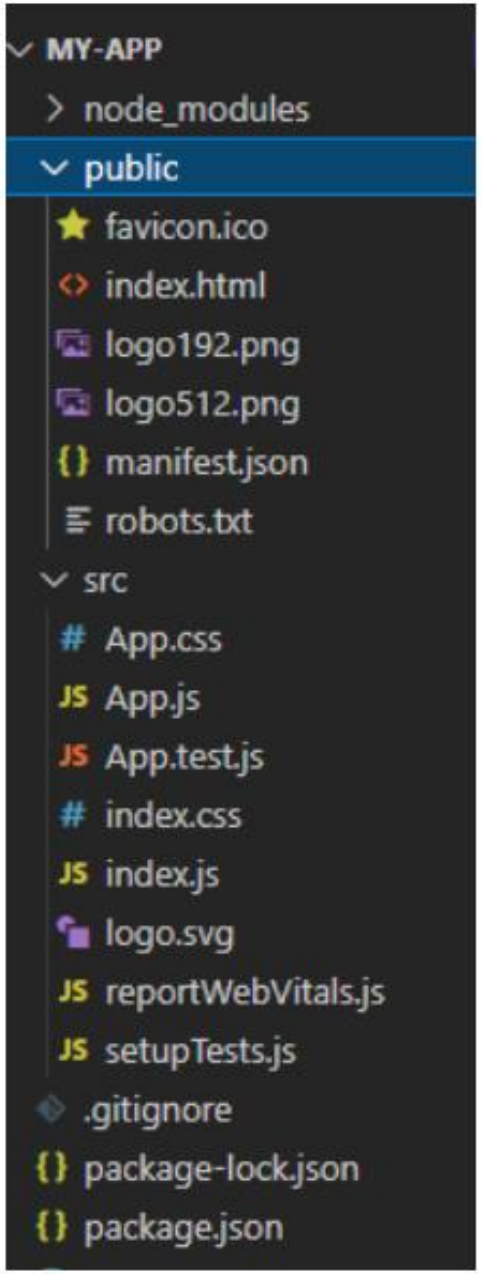
Create React App is deprecated

2. Using vite

```
# npm 7+, extra double-dash is needed:
npm create vite@latest my-app -- --template react

cd my-app
npm install
npm run dev
```





node_modules	All the node module dependencies are created in this folder
public	This folder contains the public static assets of the application
public/index.html	This is the first page that gets loaded when you run the application
src	All application related files/folders are created in this folder
src/index.js	This is the entry point of the application
package.Json	Contains the dependencies of the React application

To run the application, navigate to the folder my-app and run the command as shown below

D:/>my-app>npm start

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

ReactDOM.render is used to render an element or component to the virtual DOM.

- The first argument specifies what needs to be rendered
- The second argument specifies where to render. Observe the file **src/index.js**

The root element is present inside **index.html**

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

App.js - contains the components

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

Page Setup

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>React Samples</title>
    </head>
    <body>
        <!-- Target container -->
        <div id="root"></div>
        <!-- React library & ReactDOM (Development Version)-->
        <script
            src="https://unpkg.com/react@16/umd/react.development.js">
        </script>
        <script
            src="https://unpkg.com/react-dom@16/umd/react-dom.development.js">
        </script>
        <script>
            // Pure React and JavaScript code
        </script>
    </body>
</html>
```

Root element of the ReactDOM (starting point)

React library

ReactDOM

User interface Logic(childrens of root element)

Page Setup

```
<html>
    <head>
        <title>React Samples</title>
    </head>
    <body>
        <!-- Target container -->
        <div id="root"></div>
        <!-- React library & ReactDOM (Development Version)-->
        <script
            src="https://unpkg.com/react@16/umd/react.development.js">
        </script>
        <script
            src="https://unpkg.com/react-dom@16/umd/react-dom.development.js">
        </script>
        <script>
            // Pure React and JavaScript code
            const dish = React.createElement("h1", null, "Baked Salmon");
            ReactDOM.render(dish, document.getElementById("root"));
        </script>
    </body>
</html>
```

Root element of the ReactDOM (starting point)

React library

ReactDOM

React element

Rendering(displaying)

React Element

- React element is a plain java script object that describes the content to display on the screen by the user.
- React DOM takes care of updating the DOM to match the React elements.
- React Element can be created by createElement() or JSX syntax.

```
import React from 'react'

React. createElement(type, props, ...children)
```

returns a React element object with a few properties:

- type: The type you have passed.
- props: The props you have passed except for ref and key.
- ref: The ref you have passed. If missing, null.
- key: The key you have passed, coerced to a string. If missing, null.

Example

The user want to display on screen is “Hello, World!”

DOM element is <h1>Hello, World</h1>

React element to display Hello, World! Is as follows

```
import React from 'react';

const element = React.createElement(
                'h1',                    ──► type
                 {},                     ──► props
                'Hello, World!'          ──► children
    );
}
```

The react Object is

```
{
   "type": "h1",
   "key": null,
   "ref": null,
   "props": {
        "children": “Hello , World!"
   },
   "_owner": null,
   "_store": {}
}
```

The user want to display on screen is “Hello, {name} Welcome ”

createElement(type, props, ...children)

Creating an element without JSX

```
import { createElement } from 'react';

    function Greeting({ name }) {
        return createElement(
                'h1',
                { className: 'greeting' },
                'Hello',
                createElement('i', null, name),
                '. Welcome!'
        );
    }
```

type

props

childern

Creating an element with JSX

```
function Greeting({ name }) {
     return (
       <h1 className="greeting">
           Hello <i>{name}</i>. Welcome!
       </h1>
     );
}
```

Creating an element without JSX

```
function Greeting({ name }) {
  return createElement(
    'h1',
    { className: 'greeting' },
    'Hello ',
    createElement('i', null, name),
    '. Welcome!'
  );
}

export default function App() {
  return createElement(
    Greeting,
    { name: ‘sachin' }
  );
}
```

Creating an element with JSX

```
function Greeting({ name }) {
  return (
    <h1 className="greeting">
      Hello <i>{name}</i>. Welcome!
    </h1>
  );
}

export default function App() {
  return <Greeting name="sachin" />;
}
```

const myElement = React.createElement('h1', {}, ‘I am using JSX!');

JSX

- JSX stands for JavaScript XML or an extension of javascript .
- JSX makes it easier to write and add HTML in React.
- JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods. JSX converts HTML tags into react elements.

Example

```
const myElement = <h1>Hello, I am using JSX </h1>;

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

root.render(myElement);
```

Expressions in JSX

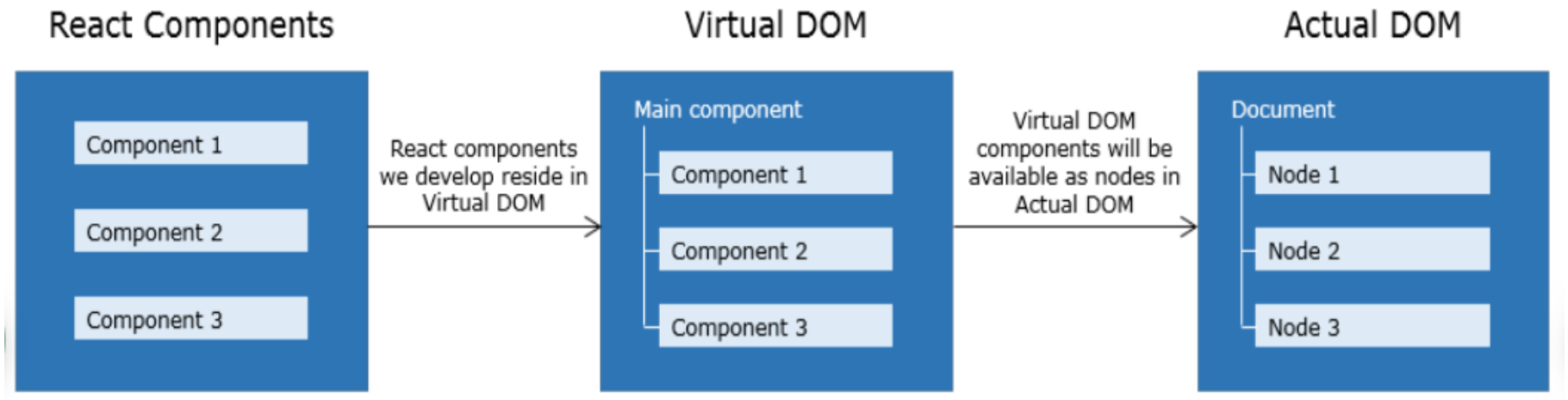
With JSX you can write expressions inside curly braces { }. The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result

Example:

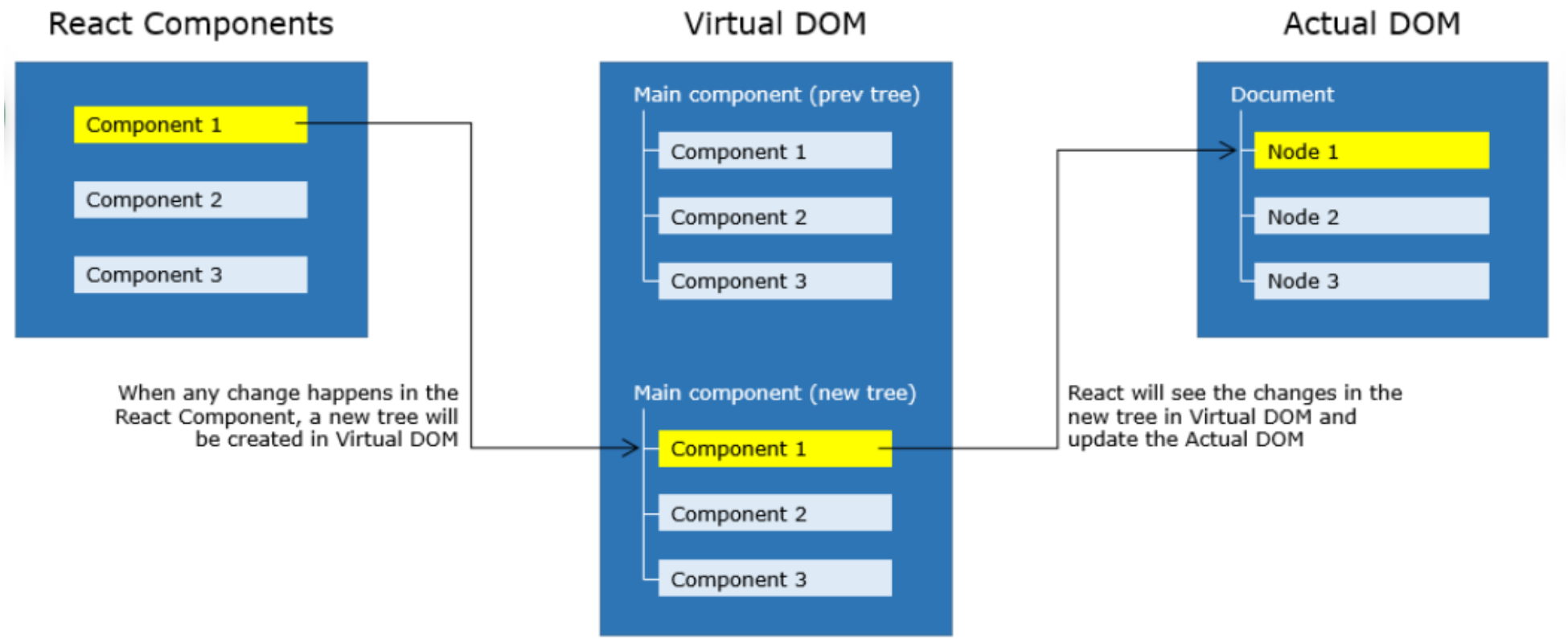
```
const myElement = <h1>React is { 10 + 5} times better with JSX</h1>;
```

React Components

- A React component is the fundamental unit of any React application that encapsulates data and view as a single unit.
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components are like javascript functions.
- In ReactJS, components reside in virtual DOM and these components will be available as nodes in actual DOM.



Virtual DOM is an abstraction of actual DOM, where components are the nodes. The virtual DOM can be programmatically modified by updating components. These updates are internally handled by React and in turn updated in actual DOM.



- Whenever there is any update in the application, the virtual DOM gets modified. React computes the difference between the previous virtual tree and the current virtual tree.
- Based on these differences React will figure out how to make updates to the actual DOM efficiently.
- React does all the computations in its abstracted DOM and updates the DOM tree accordingly.
- Virtual DOM enhances performance and efficiency by minimizing expensive updates in the actual DOM.
- Hence, React is a great performer because it manages a Virtual DOM.

React Components

Components can be created in two different ways such as:

- Functional components
- Class-based components

Function-based component

```
function MyComponent() {
      return <h1>Hello, World </h1>;
 }
```

class-based component

```
class MyComponent extends React.Component {
        render() {
           return <h1>Hello, World</h1>;
        }
  }
```

Similar to functions , a component can send data to component using props or named parameters

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

function App() {
        return (
            <div>
                <Greeting name=“Jhon" />
                <Greeting name=“Taylor" />
                <Greeting name=“Sara" />
            </div>
        );
}
```

Babel

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

Conditional Rendering

```
function Item({ name, isPacked }) {
  return <li>{name}</li>;
}

export default function PackingList() {
  return (
    <div>
      <h1>Sally Ride's Packing List</h1>
      <ul>
        <Item isPacked={true} name="Space suit" />
        <Item isPacked={true} name="Helmet with a golden leaf" />
        <Item isPacked={false} name="Photo of Tam" />
      </ul>
    </div>
  );
}
```

Conditional Rendering

```
function Course({ name, isCompleted }) {
  if (isCompleted)
      return <li>{name} ✅</li>;
  else
      return <li>{name}❌</li>;
}
export default function PackingList() {
  return (
    <div>
      <h1>Courses Completion status</h1>
      <ul>
        <Course isCompleted={true} name="Javscript" />
        <Course isCompleted={true} name="HTML and CSS" />
        <Course isCompleted={false} name="React JS" />
        <Course isCompleted={true} name="MongoDB" />
        <Course isCompleted={false} name="Angular" />
        <Course isCompleted={true} name="ExpressJS" />
      </ul>
    </div>
  );
```

Using Conditional (ternary) operator (? :)

```
function Course({ name, isCompleted }) {
    return(
          <li>
            {isCompleted ? name + " ✅ " : name + " ❌ "}
          </li>
    );
```

Using if

```
function Course({ name, isCompleted }) {
  if (isCompleted)
      return <li>{name} ✅ </li>;
  else
      return <li>{name} ❌ </li>;
}
```

Courses Completion status

- Javscript ✅
- HTML and CSS ✅
- React JS ❌
- MongoDB ✅
- Angular ❌
- ExpressJS ✅

Using Logical Operators

```
function Course({ name, isCompleted }) {
        return (<li>
        {name}{(isCompleted && '✅') || (!isCompleted && '❌')}
        </li>
        );
}
```