**React**

React is a javascript library for building user interface. It is created by Facebook developer.

React is made up of 2 libraries

* **React** library which is used to create view and it has all the code to support various component, state and props.
* **ReactDOM** as the name implies it act as a glue between React and DOM. Basically it is used to manipulate the DOM or render UI in the browser.

**Q) Why we injected React and ReactDOM?**

**A) React is present everywhere. For mobile development we use ReactNative. So, React is the core library and we use ReactDOM to work with browser.**

The simplest or smallest code for react will be to inject React CDN into the script tag(simpleCode.html). This shows that we don’t require node\_modules for injecting React into our application.

**Q) Do we need node\_modules for injecting react?**

**A) No, we can do that with React CDN**

React.createElement gives us an object which gets converted into HTML code and then puts it in the DOM. First argument will be the tag that we want to create, second argument is the prop and third argument will be the children

**React.createElement(“h1”, {}, “Welcome”);**

Tag Prop Children

**Props can be anything apart from id, class and style**.

If we create h1 element using javascript then it will be a h1 tag but in case of React it will be an object. You give element to React and it will create object out of it.

React comes with the philosophy that to manipulate the DOM with javascript.

**Starting Point**

We have to made one element as the actor or root for the react application from where it will bootstrap.

**<div id=”root”></div>**

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

**<div id=”root”>**

**<h1>This will be removed due to render()</h1>**

**</div>**

**In the above example if we do React.render then whatever was inside the root will be overridden by render()**

We will use ReactDOM single time to render UI in the browser.

If we are making the same root again then we will get warning saying react-dom.development.js:73 Warning: You are calling ReactDOMClient.createRoot() on a container that has already been passed to createRoot() before

Ex:

const heading = React.createElement("h1", {}, "Welcome to react course");

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

const root1 = ReactDOM.createRoot(document.getElementById("root"));

root.render(heading);

root1.render(heading);

Now if we are making multiple roots then React will not give warning rather it will assume multiple roots. Ideally, we should have only one root within our application where the React will be injected.

When we want to pass multiple children then in that case we should pass it as an array.

Ex: Third argument is used to put that content in the element.

        const heading1 = React.createElement("h1", {}, "Welcome");

        const heading2 = React.createElement("h1", {}, "Welcome");

        const container = React.createElement("div", {

            id: "container"

        }, [heading1, heading2]);

We can load our application directly in the production but before doing that we need to think of below points:

* Minification
* Bundling
* Optimization
* Clean console

To achieve above points we have to use bundlers. Different types of bundler available in the market are

1. Webpack
2. Parcel
3. Vite

Create-react app command uses webpack behind the scene.

To install any package in the application we use package manager by running npm init command

Npm doesn’t stand for node package manager. Since our application can’t be build just with injecting reactjs. We will need other packages and to manage/download those packages we will need npm.

Node\_modules are the db for package manager.

Generally we should not use CDN to inject any packages because our server will fetch CDN server to get those packages. Its better to have packages in our own server so that fetching time can be saved. So, mention all the packages in package.json

After adding react and react-dom in package.json we will have the package installed in node\_modules.

**How to use**

In index.html inside a script tag mention the .js file which you want to load and which has react code. But when you will launch the application browser will complain that React is not defined so the solution will be to add import statement as

import React from “react”;

import React-dom from “react-dom”;

Again browser will complain about import keyword so this time we have to tell browser that this .js is not a normal js file it a module file. How we can tell is by adding type as module in script tag as mentioned below:

**<script type=”module” src=”App.js”></script>**

**Parcel Bundler**

Things that parcel do

* HMR – Hot Module Reloading
* Clean console
* File Watcher Algorithm
* Image Optimization
* Create dev and prod builds
* Minification
* Bundling
* Caching while development
* Tree Shaking – removing unwanted code
* Port number

Polyfills is a piece of code which make our latest js code compatible with older version of browser. Babel helps to achieve this behaviour

**How JSX come into the picture**

React developer used to write React.createElement which was lengthy and if the DOM structure is complex then writing in this way very complex so Meta developer remove React from React.createElement and now the import becomes

Import { createElement } from ‘react’;

Still writing createElement() was not good then comes shorthand for createElement i.e,

Import { createElement as ce } from ‘react’;

Still it was complex then finally Meta developers created JSX file which gets converted into createElement but writing the code becomes very easy.

Ex:

Const heading = <h1 id=”heading”>Heading</h1>

This is not HTML inside Javascript. It is like HTML syntax inside Javascript.

JSX => createElement => Object => HTML(DOM)

**Babel** understands our JSX and convert into createElement

**Components in React**

* Functional Component is a function in react(Basically a arrow function)
* Class Based Component