REACT

React is a library.

LIBRARY:

FRAMEWORK:

Appendchild method is used to display an element in the browser with the help of JS

<body>

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

</body>

<script>

const heading = document.createElement("h1");

heading.innerHTML = "Hello World";

const root = document.getElementById("root")

root.appendChild(heading);

</script>

To add/inject react library to code, search react CDN

<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

React element is an object.

React has API that can be used inside HTML file to create elements, React just doesn’t have jsx way of creating elements.

Polyfill : Conversion of newer code to older code such that old versions of browser can understand it. Babel converts it.

Npx = npm + run

When we install a plugin, make sure to read how to configure it, else It will generate errors.

-Jsx is not html inside javascript, it is not HTML syntax, it is HTML like syntax. Ex JSX has camel case naming convention of attributes. Neither html or javascript understands jsx code, babel know what jsx means and its kind of a compiler, it generates a AST(Abstract syntax tree).

-JSX and react are two completely different things which are interdependent.

JSX is a modified and easier to write version of react which was developed by react developers.

Every JSX is a React element

JSX uses react.createelement.

React = 1) Elements ( ex: const will = ( <h1>Hi</h1>);)

2)Components: a) Functional component b) Class based component.

//React component

const HeadingComponent = () => {

return (

<div>

<h1>Hello</h1>

<h2>I am Saif</h2>

</div>

);

};

//React element

const header = (

<div>

<h1>What a good day!</h1>

<h3>Lets begin..</h3>

</div>

);

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

root.render(header); //way to call create element

root.render(<HeadingComponent />) //way to call react component

JSX is very secure. The JS code is first sanitized before executing it, all the APIs will be checked first to avoid XSS attack and then executed.

Js fuction is a component.

Calling of a component to be rendered is called component composition.

Jsx can have only one root/parent. We cant say/do

const test = () => {

return (

<h1>Hello</h1>

<h2>World</h2>

)

}

It will throw an error saying JSX should have one parent, hence we have to write it like this

const test = () => {

return (

<div>

<h1>Hello</h1>

<h2>World</h2>

</div>

);

};

This is called wrapping up, but so many divs can cause confusion its called React.fragment(a component of React library). react.fragment hides div(parent) from DOM to avoid confusion. <React.fragment> == <>

const test = () => {

return (

<>

<h1>Hello</h1>

<h2>World</h2>

</>

);

};

Three ways to apply CSS, 1)Inline CSS. 2)Using CSS file and 3) Using libraries(Tailwind, Bootstrap and Material UI).

<Very big deal> Config-driven UI: UI based on the certain configuration which is sent by the backend.

Optional chaining?(H/W).

Props(properties) are used alongside components to pass some data into it.

Spread operator(VV IMP): […RestaurantList].

We don’t use for loop to display all data in Industry, we use .Map method.

Virtual DOM is a software Engg concept, we keep a representation of DOM with us in our code, this is known as virtual DOM. We need virtual DOM for reconciliation in React. Reconciliation is an algorithm that React uses to differentiate one tree(Virtual DOM) from another. And it determines what needs and needs not to be changed in UI. This knowledge of changing just what needs to be changed and not the whole code(it knows that to re-render and does not render the whole code). This makes React super-fast.

To make this super fast we use something known as key, keys are assigned to divs such that react will know that in a particular component, only the key=1(say) needs to be re-render but not all the divs. We should not use Index as key.

We can just export default(default import) <component> one component from a file with the help of export default method.

To export multiple components we use export by name(Named import) method.

Local variables vs React variables: If we want our variables to be in sync with UI we have to use state variables.

A hook is just a normal function and does a particular job only.A useState is used only to create state variables.

Hooks works purely on reconciliation. Re-rendering only the selected component and not the whole website.

Synthetic events.

React only support one way binding, hence we need to use these useFucntions() to make it a two way binding.

onChange={(e) => {

setSearchText(e.target.value);

}}

For change of events we use the above way.

onClick={() => {

setSearchClicked("true") ;

}}

For simple click operations, we use this way.

To make search functionality work with present data, we need to use onChanged and onClick both of the ways of implementation.

* Smooth and good UI
* It remembers users logs
* Cookies everywhere
* Sharing with other apps
* BMI -

Microservices: Opp of monolith.

Monolith: A single project/repo, everything is in one single big project/file. Opposite of microservices. We have to deploy the whole project for even a small change.

useEffect: render something and keep updating it. Useeffect is a function that consist of callback function and a dependency array. Anything which we need to do on some dependency of a state variable, we use useeffect.

Api are called inside these useeffects with an empty dependency array.

The first render will render with initial data, aka state value or return (); then the useeffects are called.

Shimmer effect/UI:

Conditional redering:

Async await:??

Early return

Never create a component inside a component.

Never write useState inside if else, for and while(loops basically).

Use formik to create ssurvey forms.

For routing we use useRouteError, a hook to diplay errors by further destrutring the message, we use createBrowserRouter to define paths, elements,errors and make childrens related to it. We us routeProvider inside render with a prop of createBrowserRouter to render the page. We use Link to create a link of elFements in any component that needs to be called for a new URL. We use Outlet component from route to handle children inside createBrowserRouter. We use useParam to read the dynamic URL to provide unique ID’s to elements.

There are two types of routing, client side and server side. Server side implies url is updated to go to particular page from server while client side means some clicking action takes to page. We prefer client side routing.

If we use anchor tag for routing then it reloads the whole page which is a bad practice, hence we use Link provided by react-dom-route which just renders the actionable element.

For class based components, we use render() { return<jsx> } . Apart from that export, import everything will be same.

Class <className> extends React.Component. this is how we declares classes.

To access props we use this.props. to access the props. State variables are created inside a constructor.

React life cycle states that first constructor will be rendered then the return value will be rendered.

This.setState is used to make use of useState similar functionlaity.

ComponentDidMoount is used in place of useEffect in class based component.

React lifecycle: react renders a component in two phases. A render phase and a commit phase.

Render phase includes constructor and render, commit phase is where react actually modifies our DOM. In simple terms, first page renders( render phase)(shimmer effect)(pure HTML of return) and second it actually updates the DOM(commit phase)(actual mounting).

git init

git add <filename>

git status -> green status

create .gitignore

npm init -y -> package-json

npm intall -D parcel -> node modules and package-lock-json

npm install react

npm install react-dom

npm install --save-dev @babel/core

Add script “start” : “parcel ./src/index.html” then npm start.

npx parcel <entry file> -> runs the project

npm parcel build <entry file> -> builds a production version but doesn’t run.

Add “start” : “parcel <entry file>, in script then use npm start to run the project.