#Why to Learn React.

→HYPE, JOB, TREND,BUILD U.I

→ MAKES EASY TO MANAGE AND BUILD COMPLEX FRONTEND.

→ IT WAS CRETED TO SOLVE GHOST MESSAGE PROBLEM BY FACEBOOK.

#React is a library or framework?

→REACT IS A LIBRARY. THE DIFFERENCE BETWEEN LIBRARY AND FRAMEWORK IS LIKE MILITARY HAVING STRICT RULES SUCH AS NAMING CONVECTIONS , FETCHING DATA ETC... EG.. DJANGOE WHILE LIBRARY IS LIKE COOLDUDE WHICH PROVIDES YOU FREEDOM.

#Topics to learn.

→core of React(State or UI manipulation)

→component Reusability.

→Reusing of component (props)

→How to propagate change(hooks)

→Additional addon to react.

1. ROUTER(react dont have router) 2. State management(redux ,redux toolkit etc...) 3. class based -components 4.BAAD appls such as social media apps ,e commerce etc..

→React have two attachments react dom and react native

while crearting webpages react and react dom is used(working on web)
while creating mobileapps react and react native are used(working on mobile)

npx (node packet executer , directly executes packages)

1. npx create-react-app 01basic (create react app is a utility or software used to create new project)

"the above utility is very bulky so vite and parcels are recommended)

goto package.json this file contain all information about the project
"web vitals inside json measures perforamance of app"
"scripts are responsible for getting the project ready for production"

1. go inside project
2. type dir(you should see package.json)
   4.npm run start (to start the start script)
   5.npm run build

using vite

1. go to root folder
2. npm create vite@latest
3. now fill details and open package,json and see differences.
   "dev dependencies are only used during development not goes to client or for production"
4. cd to the project created above

5. type dir(you will see package.json file)
6. type npm i(pacakages which are required will be installed)
7. npm run dev(to run the project)

using 1st appraach

1. inside src folder delete unnecessary files.(setupTest, reportWebVitals,index.css, logo,app.test,app.css)
2. inside index.js (Remove unnecessaay comments and imports)
3. go inside app.js(again remove) type <h1>hy</h1> inside return
4. cd inside project
5. type npm run start

using 2nd apporach

1. again remove unnecessay files
2. u will remain with two files main.jsx, app.jsx
3. do the same thing as in approach 1
4. npm run dev

“REACT REACT KRTE HAI VARIABLES KE UPDATION PE.”

*(UI UPDATION KO REACT CONTROL KRTA HAI USING HOOKS)

→Introduction.

- React js is the most popular library which is used to make single page application (single page applications are type of applications that loads once and rest all work is achieved through J.S without reloading pages).
- React JS is basically based on components which breaks the websites into many parts or components which can then be reused.
- There is no “Router” in react , we can go from one page to another without reloading and without using router. So the react applications that you create is fast and light.
- Thunder-client is an alternative of “Postman” .It is used to test A.P.I’s
- npm is a package manager which is used to install packages inside nodejs. it is better to use npx to install packages bcz npx installs packages without storing them in memory.
- UI updation ko React control krta h.

→Creating React App.

1. npm install -g create-react-app or npx create-react-app my-app
2. To run on local host 3000 use: npm start
3. Build “npm run build” **creates a build directory with a production build of your app**. Inside the build/static directory will be your JavaScript and CSS files. Each filename inside of build/static will contain a unique hash of the file contents. This hash in the file name enables long term caching techniques.

“all components are created inside src folder”

“components are of two types class-based and function-based”

“app.js is a component and index,js is the entery point”

“node_modules store all the packages that are required to develop react app including all dependencies that is why node modulce is kept inside gitignore ”

→Props & state.

Props are used to transfer data from a parent component to a child whereas state are used to manage data inside a component itself. eg.....suppose form is a component than details which are going to be filled inside form are props && details of form such as heading etc.. are state that can be changed.

→Introducing JSX

“app.js is the main file where our code to be viewd is written”

- In react there are two components:- class-based and function based. Earlier class-based components were used but now function based components are more popular.
- The code which is written inside return(.....) is JSX.
- JSX ***stands for JavaScript XML***. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React. However there are certain changes in code such as we write “class” in html but here we write “className” also for in label tag is written as htmlFor in jsx (Note:camel case is preffered.)
- In the following code we are returning a div with className=”App”. You can only return one thing i.e suppose if you insert h1 before div then you will get error. To avoid Error you can use jsx fragment.
- Jsx Fragment:- A common pattern in ***React*** is for a component to return multiple elements. you need to enclose everthing inside “<>.....</>” .
- Babel complies JSX down to React.createElement() calls

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

//Jsx fragment
function App() {
  return (
<>
<h1>hello</h1>
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
```

```
        </a>
      </header>
    </div>
</>
  );
}

//Practice code
function App() {
let name = "Tej";
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <h2> Heloo {Tej} </h2>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
```

→Single-page application vs Multi-page application

A single-page application is an app that works inside a browser and does not require page reloading during use. You are using this type of applications every day. These are, for instance: Gmail, Google Maps, Facebook or GitHub.(Dynamically cahnge content without page reloading)

Multiple-page applications work in a “traditional” way. Every change eg. display the data or submit data back to server requests rendering a new page from the server in the browser. These applications are large, bigger than SPAs because they need to be.(requests are sned again and again)

→Named Export and Default Export.

- Named exports are useful when you need to export several values.

```
const a = "tej";
const b = "rohan";
export default b;
export{a};
export{b};


// some other file
import haha, {a,b} from './module2.mjs';
console.log(haha);
console.log(a);
```

- the default export can be imported with any name. For example:

```
const k = 12;
export default k;

// some other file
import m from "./test"; // note that we have the freedom to use import m instead of import k, because k was
console.log(m); // 12
```

- MJS files mostly belong to NodeJS by OpenJS Foundation. MJS is **the recommended filename extension of ECMAScript (JavaScript) modules used in NodeJS applications**.Node.js supports two extensions to help with this: .mjs and .cjs . **.mjs files are always ES modules, and .cjs files are always CommonJS modules**, and there's no way to override these.
- To run the module use command “node modeulename.mjs

→Creating components and understanding Props.

```
//Component code
import React from 'react'
import PropTypes from 'prop-types'

export default function Navbar(props) {
  return (
    <>
    <a className="navbar-brand" href="/">{props.title}</a>
    </>
  )

Navbar.propTypes = {
   title:PropTypes.string.isRequired,  //...is required means if you will not pass any title in app.js you
};
```

```
Navbar.defaultProps ={
title:'Set title here",
};

//App.js
import './App.css';
import Navbar from './components/Navbar';

function App() {
  return (
   <>
<Navbar   title="Textutils"/>  //if we passed a number inside title we will get error due to proptype
   </>
  );
}

export default App;
```

TailWind CSS

→React Hooks.

They let you use state and other React features without writing a class. in class based components we have to use “this.state” to set the state. But here hooks helps use to use class features without creating the class.

```
import React, {useState} from 'react'

                                                //text is a state variable will store default vaue of "
                                                 //whenever we want to update text we will use setText
                                                 //id we update text , the text will be updated without
export default function TextForm(props) {
const [text,setText] = useState("Enter text here");
text="new text";//wrong way to change text
setTexr("new text") //correct way
setText("hy");  //to update the text
  return (
    <div>
      <h1>{props.heading}-{text}</h1>
      <div className="mb-3">
      <textarea className="form-control" id="myBox" rows="8"></textarea>
      </div>
      <button className="btn btn-primary">Covert to UpperCase</button>
    </div>
  )
}
```

- This new function `useState` is the first “Hook”
- Changes the state and propagate these changes inside UI or DOM.
- Main advantage is suppose you have created a variable and you want to increment it on button click then the variable will get changed in all places without using document.getElementBy.... again and again

#Virtual DOM , React Fibre and Reconciliation

→Virtual DOM

- “createRoot” method in ReactJS creates a DOM called Virtual DOM and compares its DOM with the main DOM and updates only those things which are actually Updated in UI while Browser removes full DOM and repaints it(page reloading)
- If something is getting updated again and again frequently . Instead Of updating it again and again can we update it with the final value in order to improve performance??

→React Fibre

- “React Fibre Algorithm” is used behind the scene of virtual DOM to solve above problem by including the ability to pause, abort or resolve work as new update comes in.

→Reconciliation

- Reconciliation is an React algo used to diff one tree with another (Browser Tree and React Tree) to determine which part has to be changed similar to git .
- Diffing is performed using Keys. Keys should be stable, predictable and unique for improved performance.

#Props

- Props makes a component Reuseable for eg.. you can create a card component and can use it again and again without writing same code again and again also you can change some of the properties of card such as card title for different cards by using “props”.

#useState() counter App question solution

- useState sends all the updates in the form of batches. If one batch have same type of work then no further state is changed.

```
let [count, setCount] = useState(15);
  const addValue = () => {
    setCount (count+ 1);
    setCount( count + 1);
    setCount (count+ 1);
    setCount( count + 1);
  };
  //output after first iteration will be
  //16
```

```
let [count, setCount] = useState(15);
  const addValue = () => {
    setCount((prevCounter) => prevCounter + 1);
    setCount((prevCounter) => prevCounter + 1);
    setCount((prevCounter) => prevCounter + 1);
    setCount((prevCounter) => prevCounter + 1);
  };
  //output after first iteration will be
  //19
```

#useState() hook

- useState() hook is used to manage and manipulate the state of variables.
- So, whenever you need to keep track of something that changes over time in your component, like user input, toggling a button, or any other dynamic data, you can use `useState` to manage that state.

#useCallback()

- We use above hook for optimizing a method.
- Imagine you have a function in your component that you pass down to child components as a prop. Every time your component re-renders, this function gets recreated, even if its dependencies haven't changed. This unnecessary recreation can lead to performance issues, especially if your app has a lot of components.
- Here's where `useCallback` comes to the rescue. It memoizes the function, meaning it only recreates the function if its dependencies change.

#useEffect()

- useEffect jabhe phle bar page reload hota h tab run hota hai also agrr dependencies mai kuch bhe alteration ho to dubara se run krdeta h function ko.
- Let's say you're building a weather app. You want to fetch weather data from an API when the component loads and update the UI with that data. Here's how you can use `useEffect` in a simple way:
- The `useEffect` function runs after every render if its dependency array is empty, meaning it has no dependencies and therefore runs once when the component mounts.

#useRef()

- Imagine you have a text input field in your component and you want to focus on it automatically when the component mounts or when certain conditions are met. You can use `useRef` to create a reference to that input element and then manipulate it directly in your code.
- `useRef` is a hook in React used to create a reference to a DOM element or a value that persists across renders.

#Custom Hooks

- In React, custom hooks are JavaScript functions that let you reuse logic in your components. They are a way to extract and share stateful logic between components without duplicating code. Custom hooks follow the naming convention of starting with "use" to distinguish them from regular functions.

```
import {useEffect, useState} from "react"


function useCurrencyInfo(currency){
    const [data, setData] = useState({})
    useEffect(() => {
        fetch(`https://cdn.jsdelivr.net/gh/fawazahmed0/currency-api@1/latest/currencies/${currency}.json`)
        .then((res) => res.json())
        .then((res) => setData(res[currency]))
        console.log(data);
    }, [currency])
    console.log(data);
    return data
}

export default useCurrencyInfo;
```

#React-Router

`npm i react-router-dom`

- React Router DOM is a library for React.js applications that allows you to handle routing declaratively. In simpler terms, it helps you manage the navigation and URLs in your single-page applications (SPAs).
- Instead of loading a new webpage when a user clicks on a link, React Router DOM enables you to dynamically update what the user sees based on the URL they've visited.

→Main components are:-

- `<Link ></Link>` :- It is used as a replacement of anchor tag to prevent refreshing or reloading of full page.
- `<NavLink > </NavLink>` :- Similar to <Link /> but gives you some additional functionality such as "isActive" to track the state by matching with the url.

```
 <Link to="#" className="text-white bg-orange-700 hover:bg-orange-800 focus:ring-4 focus:ring-orange-300 fo
     Get started
</Link>

 <NavLink to="/" className={({isActive}) =>`block py-2 pr-4 pl-3 duration-200 ${isActive ? "text-orange-700
          Home
 </NavLink>
```

- `createBrowserRouter([])` :- is used to provide navigation and URL management capabilities to your application. It's a component from React Router DOM that wraps your entire application and enables you to create routes based on the URL path.
- `<ReactProvider />`

```
// const router = createBrowserRouter([
//   {
//     path: '/',
//     element: <Layout/>,
//     children: [
//       {
//         path: "",
//         element: <Home />
//       },
//       {
//         path: "about",
//         element: <About />
//       },
//       {
//         path: "contact",
//         element: <Contact />
//       }
//     ]
//   }
// ])

const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path='/' element={<Layout />}>
      <Route path='' element={<Home />} />
      <Route path='about' element={<About />} />
      <Route path='contact' element={<Contact />} />
      <Route path='user/:userid' element={<User />} />
      <Route
      loader={githubInfoLoader}
      path='github'
      element={<Github />}
       />
    </Route>
```

```
  )
)

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>,
)
```

- `<Outlet / >` :- Used to Dynamically change the components when the user directs to different URL. In below code Header and Footer components remains Fixed and Outlet acts as Dynamically Changing Element

```
import React from 'react'
import Header from './components/Header/Header'
import Footer from './components/Footer/Footer'
import { Outlet } from 'react-router-dom'

function Layout() {
  return (
    <>
    <Header/>
    <Outlet />
    <Footer />
    </>
  )
}

export default Layout
```

- `useParams()` :- You can Capture the URL'S Parameter using useParams().. it is a hook provided by React Router DOM that allows you to access parameters (or URL parameters) from the current route.

```
import React from 'react'
import { useParams } from 'react-router-dom'

function User() {
    const {userid} = useParams()
  return (
    <div className='bg-gray-600 text-white text-3xl p-4'>User: {userid}</div>
  )
}

export default User
```

- `loader` :- a "loader" typically refers to a component or element that indicates to users that content is being loaded or fetched asynchronously. It optimizes the API call and fetches data earlier. It is used with useLoaderData() Hook.

```
import React, { useEffect, useState } from 'react'
import { useLoaderData } from 'react-router-dom'

function Github() {
    const data = useLoaderData()
  return (
    <div className='text-center m-4 bg-gray-600 text-white p-4 text-3xl'>Github followers: {data.followers}
    <img src={data.avatar_url} alt="Git picture" width={300} />
    </div>
  )
}
```

```
export default Github

export const githubInfoLoader = async () => {
    const response = await fetch('https://api.github.com/users/hiteshchoudhary')
    return response.json()
}
```

#Context API

The Context API in React is used to manage global state in your application. It allows you to share data between components without having to pass props manually at every level of your component tree.

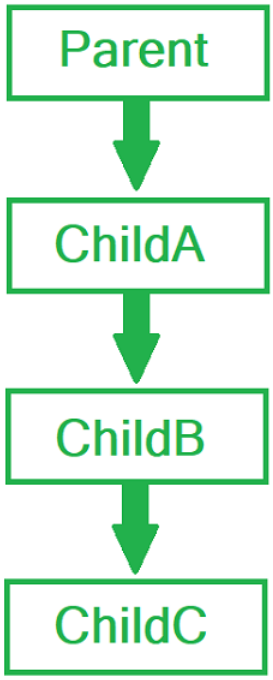
→ `createContext` in React is used to create a new context object. This context object allows you to share data across components without having to pass props manually at each level.

→ In React, the `Provider` component is used to provide data to components that need it through the Context API. Here's a simple explanation:

- **Providing Context:** The `Provider` component wraps around part of your component tree and passes down data (through a context object) to all components nested within it.
- **Sharing Data:** Components deeper in the hierarchy can then access this data without having to pass props through every level manually.
- **Usage:** You typically use `Provider` in conjunction with `createContext()` to establish a context and make its data available to other components.

→ `useContext` in React is used to consume data from a Context API that has been set up using `createContext()` and provided by a `Provider` component.

"Context Api solves the problem of prop Drilling"




```
//App.js

import './App.css';
import React , {createContext} from 'react';
import ChildA from './ChildA';

const data = createContext();
const data1 = createContext();

function App(){
   const nsme="tej"
   const gender ="Male"
   return (
      <>
      <data.Provider value={name}>
        <data1.Provider value={gender} >
          <ChildA / >
        <data1.Provider>
      <data.Provider>

      </>
   );

   export {data,data1};
```

```
//ChildC.js
import React,{userContext} from "react"
import {data,data1} from './App'

function ChildC(){
```

```
    const FirstName = useContext(data);
    const gender = useContext(data1);

    return(
       <>
         <h1>Hi my name is {FirstNamr} and my gender is {gender} </h1>
       </>
    )
    }

    export default ChildC;
```

→ReactRouter.

React Router is **a JavaScript framework that lets us handle client and server-side routing in React applications**. It enables the creation of single-page web or mobile apps that allow navigating without refreshing the page. It also allows us to use browser history features while preserving the right application view.

```
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link
} from "react-router-dom";

return (
   <>
     <Router>
      <Navbar title="Textutils" mode={mode} toggleMode={toggleMode} />
      <Alert alert={alert}/>
      <div className="container my-3">
      <Switch>
          <Route path="/about">
            <About />
          </Route>
          <Route path="/">
            <TextForm showAlert={showAlert} heading="Enter your text below to analyze." mode={mode}/>
          </Route>
      </Switch>
     </div>
     </Router>
   </>
  );


//*************************************************************************//
import React from 'react'
import { Link } from 'react-router-dom';

export default function Navbar(props) {
  return (
    <>
       <nav className={`navbar navbar-expand-lg  navbar-${props.mode}  bg-${props.mode}`}>
       <ul className="navbar-nav me-auto mb-2 mb-lg-0">
                        <li className="nav-item">
                            <Link className="nav-link" aria-current="page" to="/">Home</Link>
                        </li>
                        <li className="nav-item">
                            <Link className="nav-link" to="/about">{props.aboutText}</Link>
                        </li>
        </ul>
      </nav>
    </>
  )
}
```