Complete React.JS Tutorial In Hindi

Date: / / Page :

Topics :—

(i) Why React.js
(ii) What is React.js
(iii) Installation steps
(iv) JSX → { variable, object creation & use }
(v) What is Component, creation & Call
(vi) Reusable Components.
(vii) What is props
(viii) && operator, Ternary Operator { Conditional Rendering }
(ix) Ways to Integrate CSS
(x) Events in React { onClick, onMouseOver---}
(xi) Why useState
(xii) useState Hook
(xiii) Map in React
(xiv) Filter in React
(xv) Project - 1 [MovieZone]
(xvi) useEffect Hook
(xvii) Data Fetching from API [useEffect - jsonplaceholder]
(xviii) Form Handling
(xix) Multiple Input Handling
(xx) Project - 2 [Food Recipe]
(xxi) What is React - Router, Installation & setup
(xxii) Route Creation, Dynamic Route, Links
(xxiii) React Router Hooks - [useParams, useLocation, useNavigate]
(xxiv) Dynamic Navigation bar & Conditional Rendering.
(xxv) What is Props Drilling [Why useContext]
(xxvi) useContext Hook
(xxvii) Project - 3 [Pixabay Clone] - all above topic cover's
(xxviii) Deploy React App to vercel
(xxix) React - Redux - [useSelector, useDispatch]
(xxx) What Next ... ?

Date: / / Page:

(I) Why React.JS :-

(1) **High Demand for Jobs :-**
Numerous job opportunities for React Developers.

(2) **Backed by Facebook :-**
Reliable & well-supported.

(3) **Component-Based :-**
Build reusable parts for easy.

(4) **Virtual DOM :-**
Fast updates & rendering

(5) **JSX :-**
HTML-like code in JavaScript.

(6) **Clear Syntax :-**
Simple & understandable code.

(7) **React Native :-**
Build mobile apps with React

(8) **Server-Side Rendering :-**
Improve speed & SEO

(9) **Flexible & Scalable :-**
Suitable for all project sizes.

Date: / / Page:

(II) What is React.JS :-

(i) React.js is a popular JavaScript library/framework used for building user interfaces, especially for single-page applications.

(ii) It allows developer to create reusable components, manage the state of applications efficiently.

(iii) React has strong ecosystem provides many tools & community support.

(iv) React is Developed & maintained by Facebook.

(v) React also offer mobile app development through React Native.

Date: / / Page.

(III) Installation steps :-

(1) Install Node.js & npm (Node Package Manager)

(2) Create a folder - My React App

(3) Navigate to folder {My-React-App} & open terminal

(4) Run Command :- "npm create vite@latest"

(5) Type "." & Enter

(6) Choose a framework :- "React"

(7) Select a variant :- "JavaScript + SWC"

(8) npm install or npm i

(9) npm run dev

(10) open any browser & run :- http://localhost:5173

Date: / / Page:

(IV) JSX :-

JSX stands for JavaScript XML {extensible Markup Language}. It is a syntax extension for Javascript that allows you to write HTML-like code directly within JavaScript.

(i) HTML-Like Syntax :-

```
const element = <h1> Hello, JSX! </h1>;
```

(ii) Embedding Expression :-

It allows embedding JavaScript expressions within curly braces '{}'. This enable dynamic content rendering & logic within JSX.

```
const name = 'Suman';
const element = <h1> Hello, {name}! </h1>;
```

(iii) Babel Compilation :-

JSX is not directly understood by browser, so it need to be transpiled into regular JavaScript using Babel, which convert JSX into 'React.createElement()' calls.

JSX :-

```
const element = <h1> Hello, Suman! </h1>;
```

Transpiled by Babel into :-

JavaScript :-

```
const element = React.createElement('h1', null, 'Hello, Suman!');
```

Date: / / Page:

(V) Component Creation & call :-

(A) Component :-

In React.js, components are the building blocks that allow you to split the UI into independent & reusable pieces.

or :-

A component in React is a JavaScript function or class that optionally accepts props & returns a React element that describes how a section of the UI should appear.

(A) Types of Components :-

(i) Functional Components :-

```
import React from 'react';

const Greeting = () => {
return <h1> Hello, Suman ! </h1>; }

export default Greeting;
```

(ii) Class Components (or older style) :-

```
import React, {Component} from 'react';

class Greeting extends Component {

render() { return <h1> Hello, Suman ! </h1> ; } }

export default Greeting;
```

(VI) Reusable Components :-

(A) Person.jsx :-

```
const Person = () => {
  return (
    <>
    <h1> Hello, Suman! </h1>
    </>
  ); };
export default Person;
```

(B) App.jsx :-

```
import Person from "./Person";

  const App = () => {
  return (
    <>
    <Person />
    <Person />
    <Person />
  <>) };

export default App;
```

output :-

Hello, Suman!
Hello, Suman!
Hello, Suman!

Date: / / Page:

(VII) Poops :-

Poops is a property of component, which can be passed from one component to another or other component.

It is way to transfer data from one component to another component.

Notes:- (i) We can only pass, poops from parent Component to Child Component.

(ii) To receive poops (data) from child Component to Parent, we have to create function in parent & pass, as a poops in child.

(*) Parent Component :-

Parent Component is a Component where the another component render

(*) Child Component :-

let say we have "Person.jsx" & Person.jsx is render inside "App.jsx" then Person.jsx is known as child Component.

Note :- (*) With the help of poops we can pass :-

(a) variable → [numbers, decimals, strings, characters]
(b) objects
(c) functions
(d) Arrays, string {Anything}
(e) State.

Date: / / Page :

(3) App.jsx :- (Parent Component)

```
import React, from "React";
import Person from "./Person";

const App = () => {
return ( <>

<Person name="Suman" age={24} salary={100000}/>
<Person name="Superman" age={100} salary={500}/>
<Person name="Spiderman" age={10} Salary={1}/>
```

These are the poops {poopesty}

```
<>)} export default App;
```

(4) Person.jsx :- (Child Component)

```
import React, from "react";
```

Compulsory to use "poops" keyword

```
const Person = (poops) => {
return (<>
   <h1> My Name = {poops.name} </h1>
   <h2> My Age = {poops.age} </h1>
   <h3> My salary = {poops.salary} </h3>
  <>)
  }
export default Person;
```

(4) output :-

My Name = Suman	My Name = Superman	My name = Spiderman
My Age = 24	My Age = 100	My Age = 10
My Salary = 100000	My salary = 500	My Salary = 1

Date: / / Page:

More on Props :-

App.jsx :-

```
import React, from "react";
import Product from './Product'

const App = () => {

  const title = "iphone-17";
  const desc = {
        screen : 5.9,
        Ram : '4GB',
        ROM : '16GB',
        Brand : 'Apple' }
  const price = 75000;
return (<>
    < Product title={title} desc={desc} price={price} />
    </>)
}; export default App;
```

Product.jsx :-

```
import React, from "react";

const Product = ({title, desc, price}) => {
return (<>
    <h1> Title = {title} </h1>
    <div> <p> ScreenSize = {desc.screen} </p> </div>
    <div> <p> Ram = {desc.Ram} </p>
        <p> ROM = {desc.ROM} </p>
        <p> Brand = {desc.Brand} </p> </div>
    <h1> Price = {desc.price} </h1>
    </>
)}
```

export default Product

output :-

Title = iphone-17
ScreenSize = 5.9
Ram = 4GB
Rom = 16GB
Brand = Apple
Price = 75000

Note :- We can also pass function, array as a props from parent Component to child Component.

Date: / / Page

(VIII) && Operator, Ternary Operator & Conditional Rendering :-

(1) Ternary Operator :- ( ) ? ( ) : ( )

(A) Person.jsx :-

```
import React, from "react";

const Person = () => {
  const age = 20;
return (<>
  <div>
  <h1> { age > 18 ? (<>
                <div> You are eligible
                  for Driving License
                </div>
                </>):
          (<> <div> You are not eligible
                for Driving License </div>
                </>)}
  </h1> </div> </>) }
export default Person;
```

question mark

(A) ( ) ? ( ) : ( )

- ( ) → condition
- ( ) → In Case condition true
- ( ) → In Case Condition false

output :-

You are eligibal for Driving License

(11) && operator :-

(A) Person.jsx :-

```
import React, from "react";

const Person = () => {
  const PANCard = true;
  const gmailID = false;
return ( <>
{(PANCard) && <h1> You Can Open Bank Account </h1>}
{(gmailID) && <h1>You Can Make youTube Account</h1>}
</>)}
export default Person;
```

output :- You Can Open Bank Account

Explaation :-

&& only run, when statement true

InCase of PANCard it's true, that's why it showing
InCase of gmailID it's false, that's why it's not showing

Date: / / Page:

(IX) Ways to Integrate CSS :-

There are three ways to integrate CSS in react.js

(i) Inline
(ii) Internal
(iii) External

Note:- In React.js we have to use "className" in place of "class", because "class" is a reserve keyword in Javascript.

(i) Inline Styling :-

(A) Product.jsx :-

```
const Product = () => {
    const obj = {
        title : "iphone-17",
        price : 75000
        Ram : '8GB',
        Rom : '2GB'}

return (<>
    <div>
    <div style={{ textAlign : "center",
                  backgroundColor : "grey",
                  padding : "10px",
                  margin : "10px", }}>
<h1> title = {obj.title} </h1>
<h2> Price = {obj.price} </h2>
<h3> Ram = {obj.Ram} </h3>
<h4> Rom = {obj.Rom} </h4> </div> </div> <>) };
export default Product;
```

keyword

Date: / / Page :

(11) Internal styling :-

(4) Product.jsx :-

```
const Product = () => {

const myStyle = {

    textAlign: "center",
backgroundColor: "blue",
    padding : "20px"
    margin : "10px" }

return (<>
    <div   style = {myStyle} >

    <h1> title = Iphone-17 </h1>
    <h2> price = 75000 </h1>
    <h3> Ram = 8GB </ h3>
    <4> ROM = 2GB </h4>
    </div>
</>)
  }
export default Product;
```

Date: / / Page:

(iii) External Styling :-

In Normal - HTML ⇒ "class"
In React - JSX ⇒ "className"

(4) Product.jsx :-

```
import React, from "react";
import './product.css';
const Product = () => {
  const obj = {
      title: "Iphone",
      price: 75000,
      camera: '16 MP'
      }
return (<>
    <div className = "product_style">
    <h1> Title = {obj.title} </h1>
    <h2> Price = {obj.price} </h2>
    <h3> Camera = {obj.camera} </h3>
    </div>
    <>)
  }
export default Product;
```

(5) product.css :-

```
.product_style {

    background-color : blue;
      text-align: center;
        margin : 10px;
        padding : 20px }
```

Date: / / Page:

(x) Events in React:-

In Reactjs, events are handled similarly to how they are in vanilla JavaScript, but there are some syntatic differences.

(x) Commonly Used Events:-

(a) Form Events:- onChange, onSubmit

(b) Mouse Events:- onClick, onDoubleClick, onMouseEnter

(c) keyboard Events:- onkeyDown, onkeyUp, onkeypress

(d) Focus Events:- onFocus, onBlur

(x) App.jsx:-

```
import React from 'react';

const App = () => {

  const handleClick = () => {
    alert ('Button was clicked!'); };

  return (
      <button onClick = {handleClick}>
        Click Me
      </button>)};

export default App;
```

④ Passing Arguments to Event Handlers :-

App.jsx :-

```
import React from 'react';

const App = () => {

  const handleClick = (e) => {
    alert('Button ${e} was clicked!'); }

  return (
    <button onClick={() => handleClick(10)}>
      Click Me
    </button>)
  };

export default App;
```

Note :- If any event is taking argument, then always call that element via callback function.

(XI) Why useState :-

The normal variables are immutable in React.JS. if we try to change the variable value then it will not reflect in-to the UI.

(A) Counter.jsx :-

```
import React from 'react';
const Counter = () => {
  let count = 0;

const InCounter = () => count++;

const DeCounter = () => count--;

return ( <> <h1> {count} </h1>
    <button onClick={()=> InCounter}> InCrease </button>

    <button onclick={DeCounter}> DeCrease </button>
  <>
)
}
export default Counter;
```

(XII) **useState:-**

The 'useState' hook is a hook, that is part of the Hooks API introduced in React 16.8. 'useState' is used to declare state variable in functional Components, enabling them to hold & update, state over the time.

App.jsx:-

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

const App = () => {

const [counter, setCounter] = useState(0);
        ↓          ↓           ↓
  variable_name  function  Initial value/State

return (
  <>
  <h1> {counter} </h1>

<button onClick = {()=>setCounter(counter+1)}> Increase </button>

<button onClick = {()=>setCounter(counter-1)}> Decrease </button>

  <>
  )
  }
export default App;
```

Date: / / Page :

(XIII) **Map :-**

In React, the 'map' function is used to iterate over an array & transform each element into a new element, often a JSX element. This is very useful for rendering list of items dynamically.

App.jsx :-

```
import React from 'react';

const App = () => {

const items = ['Apple', 'Banana', 'Cherry'];

return(
    <ul>
      {items.map((item, index) =>(
          <li key={index}> {item} </li>
      ))}
    </ul>) };

export default App;
```

Note :- In the above example, 'map' is used to iterate over the 'items' array & return a list item ('<li>') for each element.

The 'key' prop is used to give each list item a unique identifier, which helps React optimize rendering.

Date: / / Page:

⑧ More on Map :-

App.jsx :-

```
import React from 'react';

const App = () => {

const products = [
    { id: 1, title: 'Sony xperia', price: 85000},
    {id:2, title: 'Iphone-16', price: 90000},
    {id:3, title: 'Galaxy S23 Ultra', price: 100000}
  ]

return ( <>
    { products.map ((data) => (
        <div key={data.id}>
        <h1>{data.title}</h1>
        <p> {data.price}</p>))
    }

    </>
    )
}
export default App;
```

Note:- To show list of Array use Map.

Date: / / Page

(XIV) Filter:-

In React, 'filter' is used to create a new array that includes only the elements that pass a specific condition.

(A) App.jsx:-

```
import React from 'react';

const App = () => {

const users = [
  {id:1, name: 'Suman', active: true},
  {id: 2, name: 'Bharti', active: false},
  {id: 3, name: 'Spiderman', active: true} ];

// Filter the users to get only the active ones

const activeUsers = users.filter(user => user.active);

return (
  <ul>
  { activeUsers.map(user => (
      <li key={user.id}> {user.name} </li>

    ))}
  </ul>
  );
};

export default App;
```

(XVI) useEffect:-

The 'useEffect' hook is used in React functional components to perform side effect in the components.

Side effects can include thing like data fetching, subscriptions, manually changing the DOM. 'useEffect' is called after the component renders & each update.

(4) App.jsx:-

```
import { useState, useEffect } from 'react';

const App = () => {

const [counter, setCounter] = useState(0);

useEffect (() => {
  document.title = counter // changing title dynamically

    console.log(counter)

}, []); // The empty array [] (dependency array)

return (<>
    <button onClick={() => counter(counter+1)} >click </button>

    </>)};
export default App;
```

Note:- If empty array [] (dependency array) is blank, then useEffect runs just onces. but if we pass something, if that thing chang then useEffect rerender again.

Date: / / Page :

XII Data Fetching from API :-

④ App.jsx :-

```
import {useState, useEffect} from 'react';

const App = () => {

const [data, setData] = useState([]);

// useEffect to fetchData from JSONPlaceholder API (Any-API)
useEffect( () => {
    const fetchData = asyn( () => {

const api = await fetch("https://jsonplaceholder.typicode.com/todos")
const result = await api.json();

    console.log(result);

setData(result) // updating the state with fetched data

fetchData();

}, []);
```

④ Note :- Sometime it will happen's that you will get two output in console, when you console logging the data {console.log(result)}, because of the React Life-Cycle methods.

XIII **Form Handling :-**

Form handling in React.js involves managing the state of form inputs & handling form submission.

A **Form.jsx :-**

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

const Form = () => {

const [name, setName] = useState("");
const [email, setEmail] = useState("");
const [Pass, setPass] = useState("");

// function to handle form submission.
    const handleSubmit = (e) => {
      e.preventDefault(); // To prevent browser default
                             behaviour
      alert("your form has been submitted");
    }

return (<>
    <form onSubmit={handleSubmit}>

Name: <input type="text" value={name}
            onChange={(e) => setName(e.target.value)} />

Email: <input type="email" value={email}
            onChange={(e) => setEmail(e.target.value)} />

Password: <input type="password" value={pass}
            onChange={(e) => setPass(e.target.value)} />
    </form> </>) }

export default Form;
```

Date: / / Page :

XIX Multiple Input Handling :-

Form.jsx :-

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

const Form = ()=>{

const [formData, setFormData] = useState({
  name: '',
  email: '',
 message: '',
  age : '',
password : ''});

// Function to handle input changes
  const handleChange = (e) => {
  const {name,value} = e.target;

    setFormData ({...formData, [name]: value});};

// Function to handleSubmit
  const handleSubmit=(e) => {
    e.preventDefault();
    alert ("your form has been submitted");}

return ( <> <form onSubmit = {handleSubmit} >

Name: <input type="text" name="name" value = {formData.name}
            onChange = {handleChange} />

Email: <input type="text" name="email" value = {formData.email
            onChange = {handleChange} />
```

Date: / / Page:

Message : <input type="text" name="message"
value = {formData.message}
onChange = {handleChange} />

Age: <input type = "text"
name = "age"
value = {formData.age}
onChange = {handleChange} />

Password: <input type = "password"
name = "password"
value = {formData.password}
onChange = {handlechange} />

</form>
<>
)
};

export default Form.

Date: / / Page :

(XXI) React- Router :-

React - Router - Dom is a library used with React.js for handling routing in web applications. It allows you to navigate between different components & pages within a single-page application without requiring a full-page reload.

(A) Basic Concepts & Components :-

(1) BrowserRouter :-

The main router component that uses the HTML5 history API to keep your UI in sync with the URL.

(2) Routes :-

A wrapper components that contains all your 'Route' components.

(3) Route :-

Defines a mapping between a URL path & the component that should be rendered.

(4) Link :-

A component that renders a navigable link to different routes in your applications.

(5) useNavigate :-

A hook that allows you to navigate programmatically within your app.

Date: / / Page:

① Installation :-

"npm install react-router-dom"

② Basic setup :-

file :- App.jsx

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

const App = () => {

  return (
      <Router>
      <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/contact" element={<contact />} />
      </Routes> </Router>
      );};

export default App;

const Home = () => <h1> Home Page </h1>;

const About = () => <h1> About Page </h1>;

const Contact = () => <h1> Contact Page </h1>;
```

Date: / / Page :

XXII Route-Creation, Dynamic-Route, Links :-

A Route-Creation :- & Links :-

App.jsx :-

```
import React from 'react';                                  Link
import {BrowserRouter as Router, Routes, Route} from 'react-router-dom'

const team = () => <h1> Team Page </h1>;
const Career = () => <h1> Career Page </h1>;
const About = () => <h1> About Page </h1>;

const App = () => {
return (
 <Router>
<nav><ul> <li> <Link to = "/" > Home </Link> </li>
          <li> <Link to = "/about" > About </Link> </li>
          <li> <Link to = "/career" > Career </Link> </li>
      </ul> </nav>

<Routes>
 <Route path = "/" element = {<Team/>} />
 <Route path = "/career" element = {<Career/>} />
 <Route path = "/about" element = {<About/>} />
</Routes>
 </Router> ); }

export default App;
```

Date: / / Page :

④ Dynamic - Routing :-

```
import React from 'react';
import Home from './Home';
import UserProfile from './UserProfile';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'
```

App.jsx :-

```
const App = () => {
  return (
    <Router>
    <Routes>
    <Route path="/" element={<Home/>}/>
    <Route path="/user/:id" element={<UserProfile/>}/>
```

/user/:id - dynamic route

```
    </Routes>
    </Router> ); };

  export default App;
```

⑤ Home.jsx :-

```
import {Link} from 'react-router-dom';

const Home = () => {
return ( <div>
<ul> <li> <Link to="/user/1"> User 1 </Link></li>
     <li> <Link to="/user/2"> User 2 </Link> </li>
     <li> <Link to="/user/3"> User 3 </Link> </li>
</ul>
</div>) }

export default Home;
```

Date: / / Page :

XXIII React - Router - Hooks - [useParams, useLocation, useNavigate] :-

① useParams :-
useParams hook use with dynamic Route Only.

② UserProfile.jsx :-

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

export const UserProfile = () => {

  const {id} = useParams();

  return (
      <h1> {id} </h1> ); }
```

③ useLocation :-
The 'useLocation' hook allows you to access the current location object, which contain information about the URL of the current route. It is very useful when you need the pathname.

App.jsx :-
```
import React from 'react';
import {useLocation} from 'react-router-dom'

const App = () => {
   const location = useLocation();
return (
    <div> Current Path: {location.pathname} </div>) }

export default App;
```

Date: / / Page :

⭐ useNavigate :-

The useNavigate hook is used for programmatically navigating to different route within the application.

App.jsx:-

```
import React from 'react';
import {useNavigate} from 'react-router-dom';

const App = () => {
  const navigate = useNavigate();

  const goToHome = () => {
    navigate ('/home'); }

  const goBack = () => {
    navigate (-1); // This will navigate to the previous page
  };

return (
    <div>
<button onClick = {goToHome} > Go to Home</button>
<button onClick = {goBack} > Go Back </button>
  </div>)
  };
export default App;
```

Date: / / Page :

XXIV Dynamic Navigation bar & Conditional Rendering :-

Navbar.jsx :-

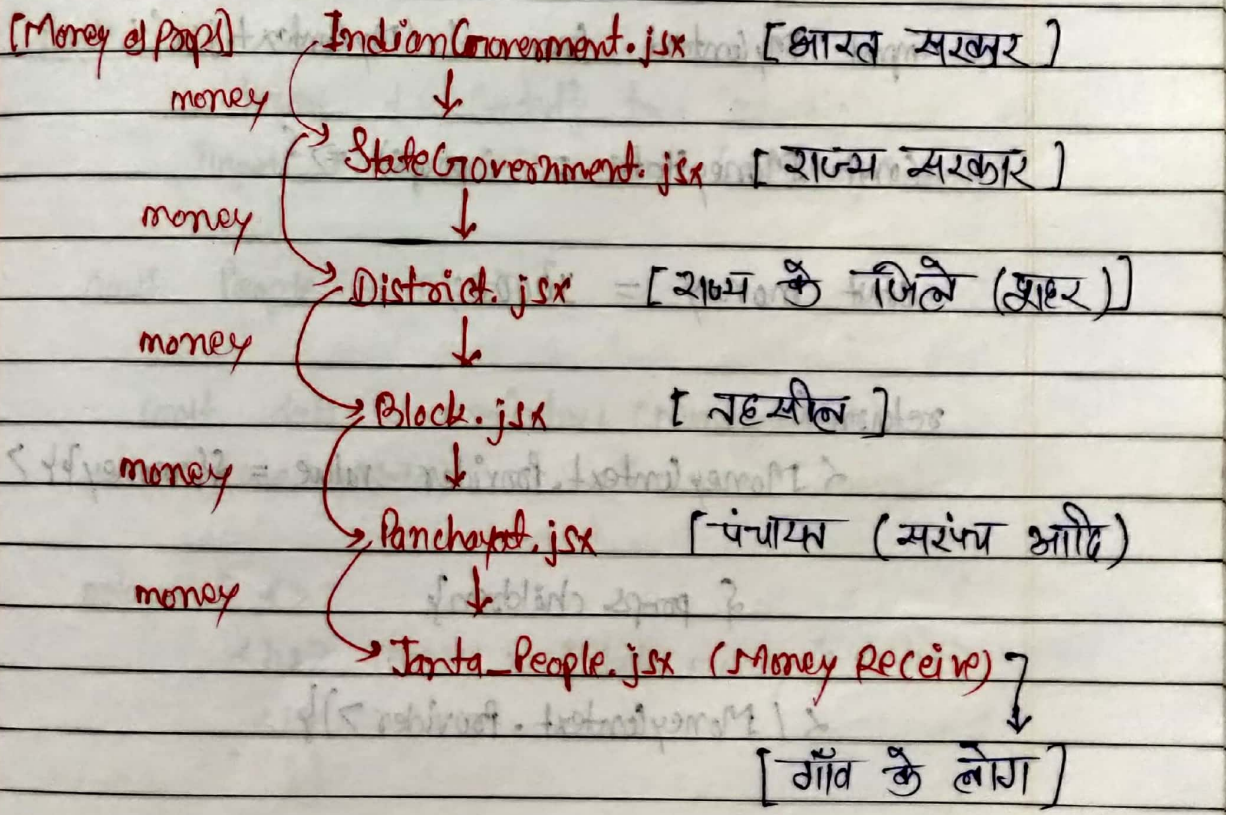
```
import React, {useState} from 'react';
import {Link, useLocation, useNavigate} from 'react-router-dom'

const Navbar = () => {
const [isLoggedIn, setIsLoggedIn] = useState(false)
  const location = useLocation();
  const Navigate = useNavigate();

const handleLogin = ()=>{
  setIsLoggedIn(true); navigate('/dashboard'); }

const handleLogOut = ()=>{
  setIsLoggedIn(false); navigate('/login'); }

return ( <div>
  <ul> <li> <Link to="/"> Home </Link> </li>
  {isLoggedIn && ( <>
      <li> <Link to="/dashboard"> Dashboard </Link> </li>
      <li> <Link to="/profile"> Profile </Link> </li>
      <li> <button onClick={handleLogOut}> LogOut </button>
      </li> </> ) }

  {!isLoggedIn && ( <>
      <li> <Link to="/about"> About </Link> </li>
      <li> <Link to="/contact"> Contact </Link> </li>
      <li> <button onClick={handleLogin}> Login </button>
      </li> </> ) }
        </div> )

export default Navbar;
```

(5) (A) **useContext :-**

The 'useContext' hook is a React hook that allows the functional components to consume values from a React context. It provides a way for components to access values from a context without having to explicitly pass them through each level of the component tree as props.

(A) **Props Drilling :-**

Suppose Indian Government wants to send some money to his janta (people) then, first he need to send money to "State Government" then "District" then "Blocks" then "Panchayat" then "Janta (people)".



भारत सरकार को अपने जनता को धन राशि भेजने के लिए इन सबसे गुजरना पड़ता था, ये था अब useContext नहीं आया था।

Date __/__/__ Page no: _____

④ Let's understands useContext :-

① Step① :- Create a Context

// MyContext.jsx :-

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

const MyContext = createContext();

export default MyContext;
```

(ii) Step② :- Create a Provider Component

④ MoneyState.jsx :-

```
import MyContext. form './MyContext.jsx';

const MoneyState = (props) => {

const money = 10000;

return (
    < MoneyContext.Provider value = {{ money }} >
        { props.children }
    </ MoneyContext.Provider >)}

export default MoneyState;
```

(iii) Step (3) :- Wrap index.jsx or App.jsx with state [MoneyState]

(a) App.jsx :-

```
import React, from 'react';
import people from './People.jsx';
import MoneyState form './MoneyState'
const APP = () => {
return ( <>
        <MoneyState>
            <People />
        </MoneyState> </> )}
export default App;
```

(iv) step (4) :- use the context (MoneyContext) in child component using useContext Hook.

(a) People.jsx :-

```
import {useContext} form 'react';
import MoneyContext from './MoneyContext'

const People = () => {
  const data = useContext (MoneyContext)
  console.log(data)
return ( <>
    <h1> Money = {data.money} </h1>
    </>
)}
export default People;
```

Redux - Toolkit

(A) Terminology :-

(i) **Redux Store :-** It is a central container that holds the entire state of application

Easy Explanation :-

Assume ki app ek Shopkeeper (dukandaar) ho, aur stockroom hai, jha saare products ko organized aur list mark rkhte ho, jisse pure shop ko easily manage kiya ja ske.

(ii) **Reducers :-** Reducers are functions in Redux, responsible for specifying how the application's state changes in response to actions. They take the current state & an action as input & return a new state.

Easy Explanation :-

Imagine shop me different departments hai, like clothing, electronics, or groceries. Each department has staff members who know how to handle updates their product category

(iii) **Slice :-**

A slice refers to piece of the application's state & the corresponding reducer logic that handles updates to that piece.

It combines a reducer function & the initial state for a specific part of the application state.

Date: / / Page no:

Easy Explanation :- Picture of different section in shop for shoes, gadgets or snacks. Each section is like a slice, and there's someone in charge of making sure products are well-stocked & organized.

(iv) useSelector :- To Select with slice want to select

(v) useDispatch :- To select with function/reducers want to use/ dispatch

Redux-toolkit in Just Four (4) Step

① **Step ①:-** Install Redux Toolkit & React-Redux

```
npm install @reduxjs/toolkit react-redux
```

② **Create a Redux Store:-**

src/redux/store/index.js

```
import {configureStore} from '@reduxjs/toolkit'

expat const store = configureStore({
    reducer: {},
});
```

③ **Step ②:-** Wrap main.jsx or index.jsx with store

main.jsx:-

```
import {store} from './redux/store'
import {Provider} from 'react-redux'

ReactDOM.createRat(document.getElementById("root")).render(

<Provider store={store}>
  <App/>
</Provider>
```

③ Step ③ :- Create a Redux state slice

Slice means function [Normal function]

⇒ redux/counterSlice/index.jsx :-

```
import { createSlice } from "@reduxjs/toolkit"

export const counterSlice = createSlice({
  name: "Counter",
  initialState: 10,
  reducers: {
      increaseby1 : (State) => State+1,
      decreaseby1 : (State) => State-1,
      increaseby10: (State) => State+10,
      decreaseby10: (State) => State-10,
  },
});

export const { increaseby1, decreaseby1, increaseby10,
      decreaseby10 } = counterSlice.actions;

export default counterSlice.reducer,
```

④ Step ① again :- Import the counterSlice in store

```
import {configureStore} from "@reduxjs/toolkit"
import counter from "../counterSlice";

export const store = configureStore({
  reducer: {
    counter: counter,
    },
  });
```

④ Step ④ :- Use Redux state & Actions in React Component

```
import { useSelector, useDispatch} from 'react-redux'
import {increaseby1, decreaseby1, increaseby10, decreaseby10}
                                    from "./redux/counterSlice"

const App = () => {
 const counter = useSelector((state) => state.counter);
 const dispatch = useDispatch();
return(
    <>
  <h1> {counter} </h1>

<button onClick={()=>dispatch(increaseby1())}>Increaseby1</button>
<button onClick={()=>dispatch(decreaseby1())}>decreaseby1</button>
<button onClick={()=>dispatch(increaseby10())}>Increaseby10</button>
<button onClick={()=>dispatch(decreaseby10())}>decreaseby10</button>
  </> )}
export default App
```