React

1. What is React ?

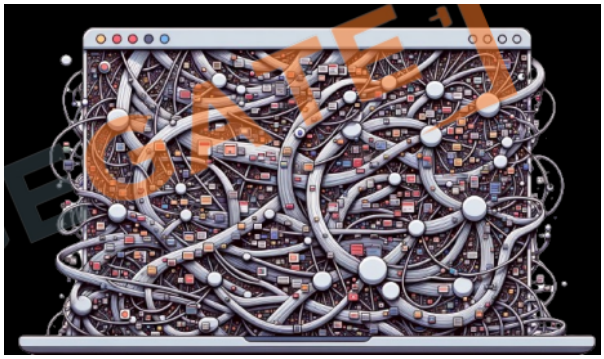
1. JavaScript library to build Dynamic and interactive user interfaces.
2. Developed at Facebook in 2011.
3. Currently the most widely used JS library for front-end development.
4. Used to create a single page application (page does not reload).

2. Working of DOM

1. Browser takes HTML and creates DOM.
2. JS helps us modify DOM based on user actions or events.
3. In big applications, Working with DOM becomes complicated.

3. Problems with JavaScript

1. React has a simpler mental model
2. JS is cumbersome
3. JS is Error-prone
4. JS is Hard to maintain

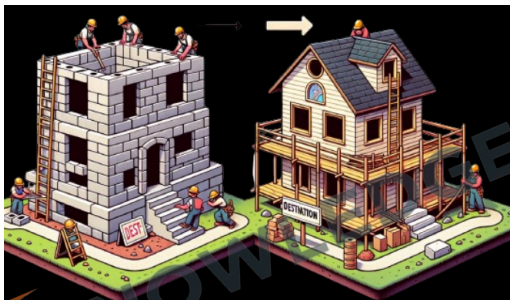


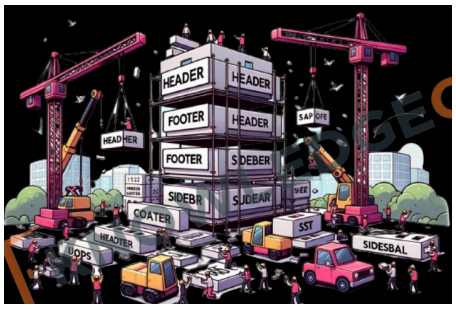
4. Working of React

1. No need to worry about querying and updating DOM elements.
2. React creates a web page with small and reusable components
3. React will take care of creating and updating DOM elements.
4. IT saves a lot of time, cheezein aasan hai, pehle se likhi hui hain.

5. JS Vs React

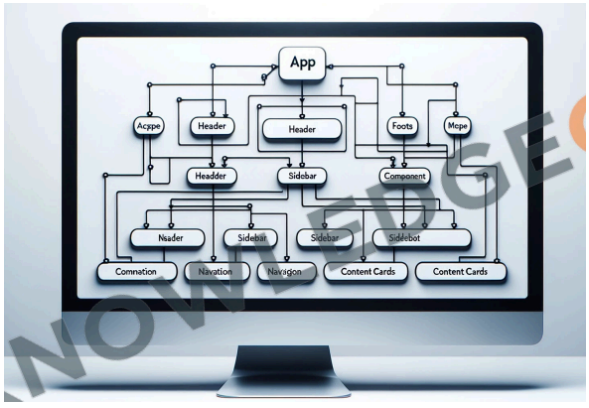
1. JS is **imperative**: You define steps to reach your desired state.
2. React is **Declarative**: You define the target UI state and then react figures out how to reach that state.





6. Introduction to Components

1. Components help us write reusable, modular and better organized code.
2. React application is a tree of components with App Component as the root bringing everything together.

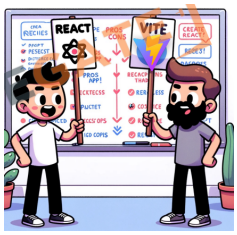


7. Install latest Node

1. Search Download NodeJS
2. Check from CMD: [**node --version**] & [**npm --version**]

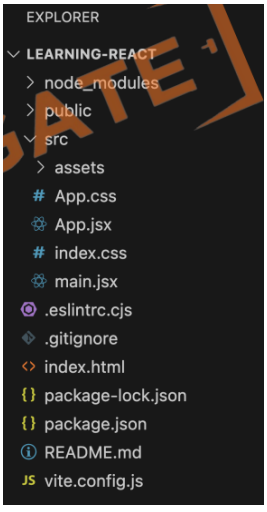
8. Create a React App

1. Official tool is CRA(Create React APP)
2. Vite is a modern tool to create React Project.
3. Vite produces Quick and Small bundle size.
4. Vite: Use npm commands (node package manager)
   npm create vite@latest [vite@4.4.1]
   cd React_Radiance
   npm install
   npm run dev [to launch dev server]
5. Use npm start for CRA.



9. Project Structure

1. **node_modules/** has all the installed node packages
2. **public/** Directory: Contains static files that don't change.
3. **src/** Directory: Main folder for the React code.
   1. **components/**: Reusable parts of the UI, like buttons or headers.
   2. **assets/**: Images, fonts, and other static files.
   3. **styles/**: CSS or stylesheets.
4. **package.json** contains information about this project like name, version, dependencies on other react packages.
5. **vite.config.js** contains vite config.



10. File Extensions

.JS

• Stands for JavaScript
• Contains regular JavaScript code
• Used for general logic and components

.JSX

• Stands for JavaScript XML
• Combines JavaScript with HTML-like tags
• Makes it easier to design UI components

11. Class vs Function Components

Class Components

• Stateful: Can manage state.
• Lifecycle: Access to lifecycle methods.
• Verbose: More boilerplate code.
• Not Preferred anymore.

Functional Components (will use this only)

• Initially stateless.
• Can use Hooks for state and effects.
• Simpler and more concise.
• More Popular.

App.jsx (example of component)

```
function App(){
  return <h1>
    Hello World!
  </h1>
}
export default App;
```

12. What is JSX?

- 1. Definition: JSX determines how the UI will look wherever the component is used.
- 2. Not HTML: Though it resembles HTML, you're actually writing JSX, which stands for JavaScript XML.
- 3. Conversion: JSX gets converted to regular JavaScript.
- 4. Babeljs.io/repl is a tool that allows you to see how JSX is transformed into JavaScript.

13. Exporting components

1. One default export: (1st way: most common)

```
function ShowBtn (){
  return <button>click me</button>
}
export default ShowBtn;

import ShowBtn from "./ComponentBtn";
function App(){
  return <div>
    <h1>
      Hello World!
    </h1>
    <ShowBtn></ShowBtn>
  </div>

}
export default App;
```

2nd way: (not preferred much)

```
export default function ShowBtn (){
  return <button>click me</button>
}
import ShowBtn from "./ComponentBtn";
```

2. Multiple named export [from one file]

```
export function ShowBtn (){
  return <button>click me</button>
}
export function Helloo (){
  return <p>My name is Prakash Jha</p>
}

import {ShowBtn, Helloo} from "./ComponentBtn";
function App(){ return <div>
    <h1>Hello World!</h1>
    <ShowBtn></ShowBtn>
    <Helloo></Helloo>
  </div>
 } export default App;
```

3. **Multiple Named export and one default export [from one file]**

```
export function ShowBtn (){
  return <button>click me</button>
}
export function Helloo (){
  return <p>My name is Prakash Jha</p>
}
export default function ShowBtn2 (){
  return <button>hit me</button>
}

import {ShowBtn, Helloo} from "./ComponentBtn";
import ShowBtn2 from "./ComponentBtn";
function App(){ return <div>
    <h1>Hello World!</h1>
    <ShowBtn></ShowBtn>
    <Helloo></Helloo>
    <ShowBtn2></ShowBtn2>
  </div>
 } export default App;
```

Exporting components:

- 1. Enables the use of a component in other parts.
- 2. Default Export: Allows exporting a single component as the default from a module.
- 3. Named Export: Allows exporting multiple items from a module.
- 4. Importing: To use an exported component, you need to import it in the destination file using import syntax.

14. Other important Points for components:

- 1. Naming: Must be capitalized; lowercase for default HTML.
- 2. HTML: Unlike vanilla JS where you can't directly write HTML, in React, you can embed HTML-like syntax using JSX.
- 3. CSS: In React, CSS can be directly imported into component files, allowing for modular and component-specific styling.

15. Dynamic Components {}

1. Dynamic Content: JSX allows the creation of dynamic and interactive UI components.
2. JavaScript Expressions: Using {}, we can embed any JS expression directly within JSX. This includes variables, function calls, and more.

```
function Helloo (){
  let name = 'Prakash';
  let fullName = () => {
    return 'Prakash Kumar Jha'
  }
  return <p>I am {name} and my full name is {fullName()}</p>
}
export default Helloo;

import Helloo from "./1stComponent";
function App(){ return <div>
    <h1>Hello World!</h1>
    <Helloo></Helloo>
  </div>
 } export default App;
```

16. Reusable Components

1. Modularity: Components are modular, allowing for easy reuse across different parts of an application.
2. Consistency: Reusing components ensures UI consistency and reduces the chance of discrepancies.
3. Efficiency: Reduces development time and effort by avoiding duplication of code.
4. Maintainability: Changes made to a reused component reflect everywhere it's used, simplifying updates and bug fixes.

```
function Random() {
  let number = Math.random()*100;

  return <h1 style={{backgroundColor: "red"}}>Random No. is
{Math.floor(number)}</h1>
}
export default Random;

import Random from "./2ndComponent";
function App(){ return <div>
    <h1>Hello World!</h1>
    <Random></Random>
    <Random></Random>
    <Random></Random>
    <Random></Random>
    <Random></Random>
  </div>
 } export default App;
```

Hello World!

Random No. is 84

Random No. is 96

Random No. is 44

Random No. is 64

Random No. is 82

17. Including Bootstrap

1. Responsive: Mobile-first design for all device sizes.
2. Components: Pre-styled elements like buttons and navbars.
3. Customizable: Modify default styles as needed.
4. Cross-Browser: Consistent look across browsers.
5. Open-Source: Free with community support.

https://getbootstrap.com/

1. Install: npm i bootstrap@5.3.3
2. Import in **main.jsx**: import "bootstrap/dist/css/bootstrap.min.css";

Simple ToDo App to show how we can use components:

```
function AppName() {
  return <h1>ToDo App</h1> }
export default AppName;
function AddTodo() {
return <div class="container text-center">
<div class="row pkj-row">
  <div class="col-6"><input type="text" placeholder="Enter ToDo
Here"/></div>
  <div class="col-4"><input type="date" /></div>
  <div class="col-2"><button type="button" class="btn btn-success
pkj-button">Add</button></div>
</div> </div> }
export default AddTodo;
function TodoItem1(){
  let todoItem ='Buy Milk';
  let todoDate = '29-10-2024';
  return <div class="container">
  <div class="row pkj-row">
    <div class="col-6">{todoItem}</div>
    <div class="col-4">{todoDate}</div>
    <div class="col-2"><button type="button" class="btn btn-danger
pkj-button">Delete</button></div>
  </div>  </div> }
export default TodoItem1;
import AppName from "./components/AppName";
import AddTodo from "./components/AddTodo";
import TodoItem1 from "./components/TodoItem1";
import TodoItem2 from "./components/TodoItem2";
import "./App.css";
function App() {
   return <center className="todo-container">
    <AppName></AppName>
    <AddTodo></AddTodo>
    <div className="items-container">
    <TodoItem1></TodoItem1>
    <TodoItem2></TodoItem2>
    </div>
  </center>}
export default App
```

18. Fragments

1. **What**? Allows grouping of multiple elements without extra DOM nodes.

2. **Why**?

• Return multiple elements without a wrapping parent.

• Cleaner DOM and consistent styling.

Error: You can't return two/more than two tags.

```
function App() {
 return <h1>hello</h1>
      <p>PPPP</p>
}
export default App;
```

Solution: div me daal do [pr ye accha tarika nhi h]

```
<div>
return <h1>hello</h1>
        <p>PPPP</p>
</div>
```

As a solution will use Fragments:

Method 1:

```
import React from "react";
function App() {
  return( <React.Fragment>
          <h1>hello</h1>
          <p>PPPP</p>
          </React.Fragment>);
}
export default App;
```

Method 2:

```
function App() {
  return(<>
          <h1>hello</h1>
          <p>PPPP</p>
          </>);
}
export default App;
```

19. Map Method (Important)

1. Purpose: Render lists from array data.
2. JSX Elements: Transform array items into JSX.
3. Inline Rendering: Directly inside JSX
{items.map(item => <li key={item.id}>{item.name})}
4. Key Prop: Assign unique key for optimized re-renders.
 {item.name}

```
function App() {
  return <>
          <h1>Healthy Foods</h1>
          <ul className="list-group">
           <li className="list-group-item">Dal</li>
           <li className="list-group-item">Roti</li>
           <li className="list-group-item">Milk</li>
           <li className="list-group-item">Ghee</li>
         </ul>
</>
}
export default App;
```

```
function App() {
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  return <>
          <h1>Healthy Foods</h1>
          <ul className="list-group">
           {foodItems.map((item) => (
             <li key={item} class="list-group-item">{item}</li>
           ))}
         </ul>
</>
}
export default App;
```

Note: JSX me hmlog loop nhi return kr skte isliye iska alternative map use kr liye to print list of items.

20. Conditional Rendering(alternative of if & else)

• Displaying content based on certain conditions.

• Allows for dynamic user interfaces.

Methods

• If-else statements: Choose between two blocks of content.

• Ternary operators: Quick way to choose between two options.

• Logical operators: Useful for rendering content when a condition is true.

Benefits

• Enhances user experience.

• Reduces unnecessary rendering.

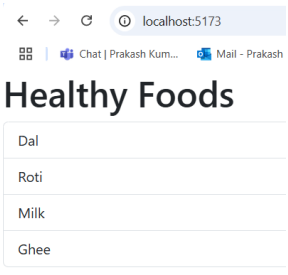
• Makes apps more interactive and responsive.

What is the **need** of it ?

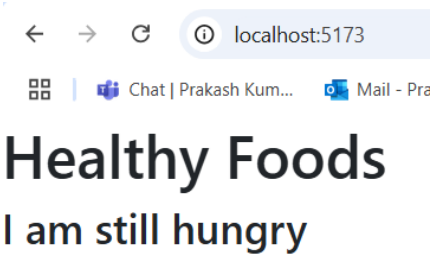
```
function App() {
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  if (foodItems.length() === 0}
    return <h1> Hello </h1>
 else
    return <h2> Hey <h2>
export default App;
```

```
Above one is possible but below one is not Possible, that's why we will
use conditional rendering. (return ke andar se if else karne ke liye)
function App() {
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
    return if (foodItems.length() === 0}
            return <h1> Hello </h1>
            else
            return <h2> Hey <h2>
export default App;
```

1.



```
function App() {
  //let foodItems = [];
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  let emptyMessage =
  foodItems.length ===0 ? <h3>I am still hungry</h3> : null;
  return <>
         <h1>Healthy Foods</h1>
         {emptyMessage}
         <ul className="list-group">
          {foodItems.map((item) => (
            <li key={item} className="list-group-item">{item}</li>
          ))}
        </ul>
</>
}
export default App;
```



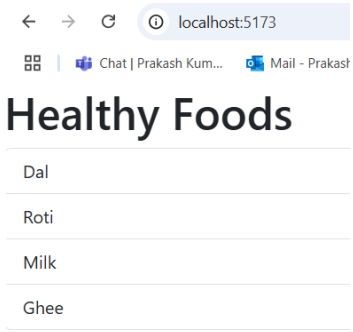
2.

```
function App() {
  let foodItems = [];
  //let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  let emptyMessage =
  foodItems.length ===0 ? <h3>I am still hungry</h3> : null;
  return <>
         <h1>Healthy Foods</h1>
         {emptyMessage}
         <ul className="list-group">
          {foodItems.map((item) => (
            <li key={item} className="list-group-item">{item}</li>
          ))}
        </ul>
</>
}
export default App;
```

3. Using logical operator

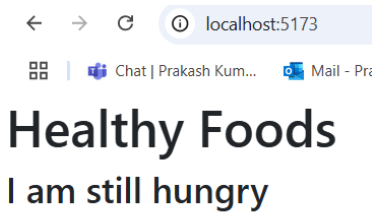
1.

```
function App() {
  //let foodItems = [];
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  return <>
         <h1>Healthy Foods</h1>
         {foodItems.length ===0 && <h3>I am still hungry</h3>}
         <ul className="list-group">
          {foodItems.map((item) => (
            <li key={item} className="list-group-item">{item}</li>
          ))}
        </ul>
</>
}
export default App;
```

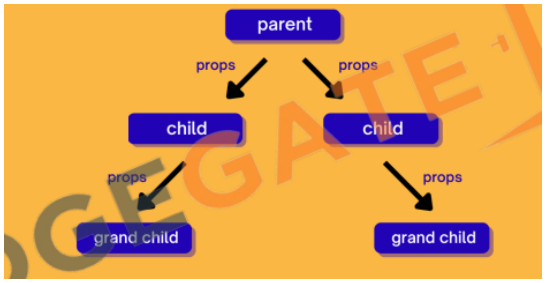


2.

```
function App() {
  let foodItems = [];
  //let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  return <>
         <h1>Healthy Foods</h1>
         {foodItems.length ===0 && <h3>I am still hungry</h3>}
         <ul className="list-group">
          {foodItems.map((item) => (
            <li key={item} className="list-group-item">{item}</li>
          ))}
        </ul>
</>
}
export default App;
```



21. Passing Data via Props (Important)



Props in React

- Short for properties
- Mechanism for passing data.
- Read-only by default Usage
- Pass data from parent to child component.
- Makes components reusable.
- Defined as attributes in JSX.

Key Points

- Data flows one-way (downwards).
- Props are immutable.
- Used for communication between components.

As a attribute value pass kr dete hain jo v krna hota hai:

Yahan foodItems pass kr rahe jisko receive kiya jayega products ke name se

```
<ErrorMessage products={foodItems}></ErrorMessage>
<FoodItems products={foodItems}></FoodItems>
```

Hm jo bhi value pass krte hain wo props ke terms me store ho jata hai.

Destructuring krke products me le aaye.

```
let FoodItems = (props) => {
  let {products} = props;
```

Sab Kuchh parent to child flow krta hai.

APP.jsx

```
import FoodItems from "./components/FoodItems";
import ErrorMessage from "./components/ErrorMessage";
function App() {
  //let foodItems = [];
  let foodItems = ["Dal", "Roti", "Milk", "Ghee"];
  return (<>
         <h1>Healthy Foods</h1>
         <ErrorMessage products={foodItems}></ErrorMessage>
         <FoodItems products={foodItems}></FoodItems>
</>
  );
}
export default App;
```

FoodItems.jsx

```
import Item from "./Item";
let FoodItems = (props) => {
  let {products} = props;
 return (<ul className="list-group">
 {products.map((item) => (
   <Item key={item} foodItem={item}></Item>
 ))}
</ul>
 ); }
export default FoodItems;
```

Item.jsx

```
let  Item = (props) => {
  let {foodItem} = props;
 return (<li className="list-group-item">{foodItem}</li> );
};
export default Item;
```

ErrorMessage.jsx

```
let ErrorMessage = (props) =>{
  let {products} = props;
  return <>{products.length ===0 && <h3>I am still hungry</h3>} </>;
};
export default ErrorMessage;
```

22. CSS Modules

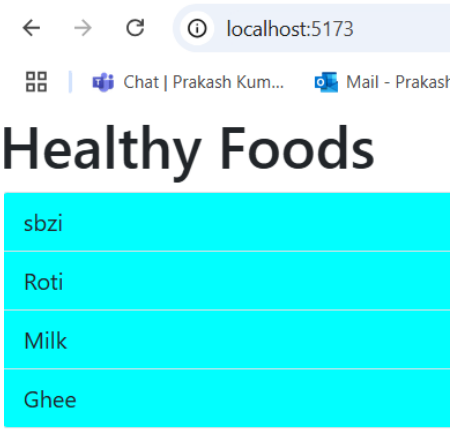
1. Localized class names to avoid global conflicts. **Jisse dusre css file me v same name se kuchh defined ho to bhi koi problem na ho.**
2. Styles are scoped to individual components.
3. Helps in creating component-specific styles.
4. Automatically generates unique class names.
5. Promotes modular and maintainable CSS.
6. Can be used alongside global CSS when needed.

Item.jsx (Not working properly)

```
import css from "./Item.module.css"
let  Item = (props) => {
  let {foodItem} = props;
 return (<li className={`${css.itemBg} list-group-item`}>{foodItem}
</li> );
};
export default Item;
```

Item.module.css

```
.itemBg{
  background-color: aqua;
}
```



Project: TODO App UI [Version:2]

```
import AppName from "./components/AppName";
import AddTodo from "./components/AddTodo";
import TodoItems from "./components/TodoItems";
import "./App.css";
function App() {
  const todoItems = [
    {name: "Buy Milk", dueDate: "4/10/2023",},
    {name: "Go to College",dueDate: "4/10/2023",},
    {name: "Like this video", dueDate: "right now",}, ];
  return (
    <center classNAme="todo-container">
      <AppName />
      <AddTodo />
      <TodoItems todoItems={todoItems}></TodoItems>
    </center> );}
export default App;//App.jsx

input {width: 100%;}
.kg-button { min-width: 80px;}
.kg-row { margin: 10px 5px;} //App.css

import styles from "./AppName.module.css";
function AppName() {
  return <h1 className={styles.todoHeading}>TODO App</h1>; }
export default AppName;

function TodoItem({ todoName, todoDate }) {
  return (
    <div className="container">
      <div className="row kg-row">
        <div className="col-6">{todoName}</div>
        <div className="col-4">{todoDate}</div>
        <div className="col-2">
          <button type="button" className="btn btn-danger kg-button">
            Delete
          </button>
        </div>
      </div>
    </div> ); } export default TodoItem;
```

```
.itemsContainer {
  text-align: left;
}//TodoItems.module.css

import TodoItem from "./TodoItem";
import styles from "./TodoItems.module.css";
const TodoItems = ({ todoItems }) => {
  return (
    <div className={styles.itemsContainer}>
      {todoItems.map((item) => (
        <TodoItem key = {item.name} todoDate={item.dueDate}
todoName={item.name}></TodoItem>
      ))}
    </div> ); };
export default TodoItems;

.todoHeading {
  font-weight: 700; font-size: 45px;
  margin: 10px; margin-bottom: 20px;
} //AppName.module.css

function AddTodo() {
  return (
    <div className="container text-center">
      <div className="row kg-row">
        <div className="col-6">
          <input type="text" placeholder="Enter Todo Here" />
        </div>
        <div className="col-4">
          <input type="date" />
        </div>
        <div className="col-2">
          <button type="button" className="btn btn-success kg-button">
            Add
          </button>
        </div>
      </div>
    </div> ); } export default AddTodo;
```

23. Passing Children

Tb use krte hain jb hmlog kisi component ka design taiyar kar rahe pr content kya hoga ye nahi pta. (Bhot mst chij h ye **)

1. children is a special prop for passing elements into components.
2. Used for flexible and reusable component designs.
4. Accessed with props.children.
5. Can be any content: strings, numbers, JSX, or components.
6. Enhances component composability and reusability.

```
import FoodItems from "./components/FoodItems";
import ErrorMessage from "./components/ErrorMessage";
import Container from "./components/Container";
function App() {
  //let foodItems = [];
  let foodItems = ["sbzi", "Roti", "Milk", "Ghee"];
  return (<>
         <Container>
          <h1>Healthy Foods</h1>
          <ErrorMessage products={foodItems}></ErrorMessage>
          <FoodItems products={foodItems}></FoodItems>
          </Container>
          <Container>
           <p>Above is the List of healthy food</p>
          </Container>
          </> ); } export default App;//App.jsx
```

```
import styles from "./Container.module.css";
let Container = (props) => {
  return <div className={styles.conDesign}>{props.children}</div>
}; export default Container;//Container.jsx
←                    OR                              →
import styles from "./Container.module.css";
let Container = ({children}) => {
  return <div className={styles.conDesign}>{children}</div>
};
export default Container;//Container.jsx
```

```
.conDesign{ border: 1px solid black; margin: 15px;//Container.module.css
 width: 50%; border-radius: 10px; min-width: 300px; padding: 15px; }
```

24. Handling Events (Important)

1. React events use camelCase, e.g., onClick.
2. Uses synthetic events, not direct browser events.
3. Event handlers can be functions or arrow functions.
4. Use onChange for controlled form inputs.
5. Avoid inline arrow functions in JSX for performance.

App.jsx

```
import FoodInput from "./components/FoodInput";
<FoodInput></FoodInput>
```

FoodInput.jsx

```
import styles from "./FoodInput.module.css";
let FoodInput = () => {
 const handleOnChange = (event) => {
  console.log(event.target.value);
 }
return <input type="text" placeholder="Enter Food Item Here"
 className={styles.food}
 onChange={handleOnChange}
 ></input>
} export default FoodInput;
```

FoodInput.module.css

```
.food{ width: 100%; padding: 5px;
  margin-top: 10px; margin-bottom: 10px; }
```

Item.jsx

```
import css from "./Item.module.css"
let  Item = (props) => {
  let {foodItem} = props;
 return (<li className={`${css["itemBg"]} list-group-item`}>{foodItem}
          <button className={`${css.button} btn btn-info`}onClick={() =>
console.log(`Buy ${foodItem}`)}>Buy</button>
          </li> ); }; export default Item;
```

Item.module.css

```
.itemBg{ background-color: aqua; }
.button{ float:right; }
```

25. Passing Function via Props

1. Pass dynamic behavior between components.
2. Enables upward communication from child to parent.
3. Commonly used for event handling.
4. Parent defines a function, the child invokes it.
5. Enhances component interactivity.

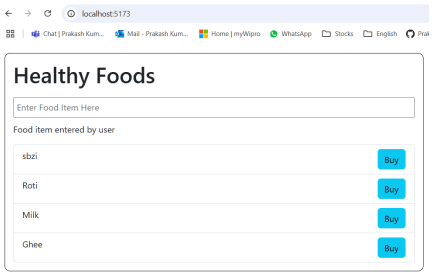
```
import Item from "./Item";
let FoodItems = ({products}) => {
 return (<ul className="list-group">
 {products.map((item) => (
   <Item key={item} foodItem={item}
   handleBuyButton={() => console.log(`${item}`)}
   ></Item> ))}
</ul> );
}export default FoodItems; //FoodItems.jsx
```

```
import css from "./Item.module.css"
let  Item = ({foodItem, handleBuyButton}) => {
 return (<li className={`${css["itemBg"]} list-group-item`}>{foodItem}
          <button className={`${css.button} btn
btn-info`}onClick={handleBuyButton}>Buy</button>
          </li> );
};
export default Item;//Items.jsx
```

Items bhi hm hi de rhe aur uska behavior kaisa hoga ye bhi hmhi bta rhe.

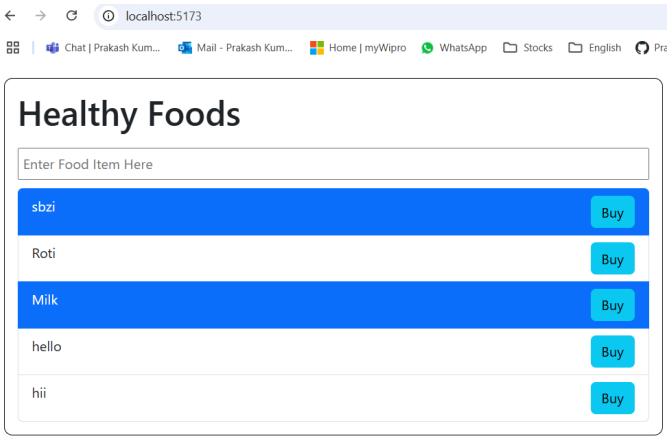
26. Managing State (Important)

1. State represents data that changes over time.
2. State is local and private to the component.
3. State changes cause the component to re-render.
4. For functional components, use the useState hook.
5. React Functions that start with word use are called hooks
6. Hooks should only be used inside components
7. Parent components can pass state down to children via props.
8. Lifting state up: share state between components by moving it to their closest common ancestor.



App.jsx

```
import FoodItems from "./components/FoodItems";
import ErrorMessage from "./components/ErrorMessage";
import Container from "./components/Container";
import FoodInput from "./components/FoodInput";
import { useState } from "react";
function App() {
  //let foodItems = [];
  let foodItems = ["sabzi", "Roti", "Milk", "Ghee"];
  // let textStateArr = useState("Food item entered by user");
  // let textToShow = textStateArr[0]; //initial value
  // let setTextState = textStateArr[1];
let [textToShow, setTextState]= useState("Food item entered by user");
 const handleOnChange = (event) => {
    setTextState(event.target.value);}
  return (<>
        <Container>
         <h1>Healthy Foods</h1>
         <ErrorMessage products={foodItems}></ErrorMessage>
         <FoodInput handleOnChange={handleOnChange}></FoodInput>
         <p>{textToShow}</p>
         <FoodItems products={foodItems}></FoodItems>
         </Container>
         <Container>
          <p>Above is</p>
         </Container>
         </> );} export default App;
import styles from "./FoodInput.module.css";
let FoodInput = ({handleOnChange}) => {
 return <input type="text" placeholder="Enter Food Item Here"
 className={styles.food}
 onChange={handleOnChange}
 ></input> } export default FoodInput;//FoodInput.jsx
```



App.jsx

```
import FoodItems from "./components/FoodItems";
import ErrorMessage from "./components/ErrorMessage";
import Container from "./components/Container";
import FoodInput from "./components/FoodInput";
import { useState } from "react";
function App() {
 let [foodItems, setFoodItems] = useState(["sbzi", "Roti", "Milk"]); //hooks ka use kr rhe states save krne ke liye
 const onKeyDown = (event) => { //koi bhi key press hone pe
  if (event.key === 'Enter'){  //jaise hi enter press ho
   let newFoodItem = event.target.value;
   event.target.value="";
   let newItems = [...foodItems, newFoodItem]; //purana array + new wala item
   setFoodItems(newItems);
  }
 }
 return (<>
     <Container>
      <h1>Healthy Foods</h1>
      <FoodInput handleKeyDown={onKeyDown}></FoodInput>
      <ErrorMessage products={foodItems}></ErrorMessage>
      <FoodItems products={foodItems}></FoodItems>
      </Container>
      </>
 );
}
export default App;

// foodItems = initial value jo hmlog pass kr rhe("sbzi", "Roti", "Milk")
//setFoodItems = state change ke baad wala
//onKeyDown = koi bhi key press hone pe
```

FoodInput.jsx

```
import styles from "./FoodInput.module.css";
let FoodInput = ({handleKeyDown}) => {
 return <input type="text" placeholder="Enter Food Item Here"
 className={styles.food}
 onKeyDown={handleKeyDown}
 ></input>
} export default FoodInput;
```

Item.jsx

```
import css from "./Item.module.css"
let  Item = ({foodItem, bought, handleBuyButton}) => {
 return (<li className={`${css["itemBg"]} list-group-item ${bought &&
'active'}`}>{foodItem} //agar value true ho to class active use ho jana
          <button className={`${css.button} btn
btn-info`}onClick={handleBuyButton}>Buy</button>
          </li> );
};
export default Item;
```

FoodItems.jsx

```
import Item from "./Item";
import { useState } from "react";
let FoodItems = ({products}) => {
  let [activeItems, setActiveItems] = useState([]); //diff state ke liye
  let onBuyButton = (item, event) => {
    let newItems = [...activeItems, item]; //array + new item
    setActiveItems(newItems);
  }
 return (<ul className="list-group">
 {products.map((item) => (
   <Item key={item} foodItem={item} bought={activeItems.includes(item)}
   handleBuyButton={(event) => onBuyButton(item, event)}
   ></Item>
 ))}  //agar item already available hai to bought true ho jaye
</ul>
 );
}export default FoodItems;
```

27. State vs Props

State:

• Local and mutable data within a component.

• Initialized within the component.

• Can change over time.

• Causes re-render when updated.

• Managed using useState in functional components.

Props:

• Passed into a component from its parent.

• Read-only (immutable) within the receiving component.

• Allow parent-to-child component communication.

• Changes in props can also cause a re-render.

Note: To import any component Place your cursor at the end of the component name. Then use —> (ctrl + space) then press ENTER

Project: Calculator

App.jsx

```
import Display from "./components/Display";
import ButtonsContainer from "./components/ButtonsContainer";
import styles from "./App.module.css";
import { useState } from "react";
function App() {
  const [calVal, setCalVal] = useState(""); //state save krne ke liye
  const onButtonClick = (buttonText) => { //buttonName ko as buttonText receive kiya
    if (buttonText === "C") { setCalVal("");
    } else if (buttonText === "=") {
      const result = eval(calVal); setCalVal(result);
    } else {
      const newDisplayValue = calVal + buttonText;
      setCalVal(newDisplayValue); }};
return (
    <div className={styles.calculator}>
      <Display displayValue={calVal}></Display> //calVal value pass kiye
      <ButtonsContainer onButtonClick={onButtonClick}></ButtonsContainer>
    </div> //onButtonClick method pass kiye
  ); }export default App;
```

App.module.css

```
.calculator {
  border: 1px solid rgba(90, 90, 90);
  width: 200px; border-radius: 5px; }
```

Display.jsx

```
import styles from "./Display.module.css";
const Display = ({ displayValue }) => { //yahan calVal receive kiye
  return (
    <input
      className={styles.display}
      type="text"
      value={displayValue}  //value me whi set kr diye
      readOnly
    />
  ); }; export default Display;
```

Display.module.css

```
.display { margin: 10px; width: 175px; font-size: 25px; }
```

ButtonContainer.jsx

```
import styles from "./ButtonsContainer.module.css";
const ButtonsContainer = ({ onButtonClick }) => {
  const buttonNames = ["C","1","2","+","3","4","-","5",
    "6","*","7","8","/","=","9","0","." ];
  return (
    <div className={styles.buttonsContainer}>
      {buttonNames.map((buttonName) => (
        <button
          className={styles.button}
          onClick={() => onButtonClick(buttonName)}
        >{buttonName} </button>))} //method receive hua aur ye buttonName return kr diya
    </div> );}; export default ButtonsContainer;
```

ButtonContainer.module.css

```
.buttonsContainer {
  display: flex; justify-content: center; flex-wrap: wrap; }
.button {
  width: 45px; height: 45px; margin: 3px; }
```

** onButtonClick method receive krke dubara se apne body se same method call kr de rha with parameter (buttonName)

Project: ToDo App (version:3/Final)

App.jsx

```
import AppName from "./components/AppName";
import AddTodo from "./components/AddTodo";
import TodoItems from "./components/TodoItems";
import WelcomeMessage from "./components/WelcomeMessage";
import "./App.css";
import { useState } from "react";
function App() {
  const [todoItems, setTodoItems] = useState([]); //to set state
  const handleNewItem = (itemName, itemDueDate) => {
    console.log(`New Item Added: ${itemName} Date:${itemDueDate}`);
    const newTodoItems = [
      ...todoItems,{ name: itemName, dueDate: itemDueDate },];//spread operator
    setTodoItems(newTodoItems);};//method body & child se 2 input expect kr rhe
  const handleDeleteItem = (todoItemName) => {
  const newTodoItems = todoItems.filter((item) => item.name !==
   todoItemName); //ye condition satisfy karne wala new array bnao
setTodoItems(newTodoItems); };//method body & child se 1 input expect kr rhe
  return (
    <center className="todo-container">
      <AppName />
      <AddTodo onNewItem={handleNewItem} /> //AddTodo child ko method pass kr rhe
      {todoItems.length === 0 && <WelcomeMessage></WelcomeMessage>}
      <TodoItems
        todoItems={todoItems} //props pass kr rhe
        onDeleteClick={handleDeleteItem}//method pass kr rhe
      ></TodoItems>
    </center> ); } export default App;
```

App.css

```
input { width: 100%; }
.kg-button { min-width: 80px; }
.kg-row { margin: 10px 5px; }
```

AppName.jsx

```
import styles from "./AppName.module.css";
function AppName() {
  return <h1 className={styles.todoHeading}>TODO App</h1>;
}export default AppName;
```

AppName.module.css

```
.todoHeading { font-weight: 700; font-size: 45px;
  margin: 10px; margin-bottom: 20px; }
```

AddToDo.jsx

```
import { useState } from "react";
function AddTodo({ onNewItem }) {
  const [todoName, setTodoName] = useState("");
  const [dueDate, setDueDate] = useState("");
  const handleNameChange = (event) => {
    setTodoName(event.target.value); };
  const handleDateChange = (event) => {
    setDueDate(event.target.value); };
  const handleAddButtonClicked = () => {
    onNewItem(todoName, dueDate); //parent ko 2 chij return kr rha
    setDueDate(""); setTodoName(""); };
  return (
    <div className="container text-center">
      <div className="row kg-row">
        <div className="col-6">
          <input
            type="text"
            placeholder="Enter Todo Here"
            value={todoName}
            onChange={handleNameChange}
          />
        </div>
        <div className="col-4">
          <input type="date" value={dueDate} onChange={handleDateChange}/>
        </div>
        <div className="col-2">
          <button
            type="button"
            className="btn btn-success kg-button"
            onClick={handleAddButtonClicked}>Add
          </button>
        </div>
      </div>
    </div>
  );}export default AddTodo;
```

TodoItems.jsx

```
import TodoItem from "./TodoItem";
import styles from "./TodoItems.module.css";
const TodoItems = ({ todoItems, onDeleteClick }) => {//parent se receive kiya
  return (
    <div className={styles.itemsContainer}>
      {todoItems.map((item) => (
        <TodoItem
          todoDate={item.dueDate}
          todoName={item.name}
          onDeleteClick={onDeleteClick} //child ko simply pass kiya
        ></TodoItem>
      ))} </div> ); }; export default TodoItems;
```

TodoItems.module.css

```
.itemsContainer { text-align: left;}
```

TodoItem.jsx

```
function TodoItem({ todoName, todoDate, onDeleteClick }) {//parent se receive kiya
  return (
    <div className="container">
      <div className="row kg-row">
        <div className="col-6">{todoName}</div>
        <div className="col-4">{todoDate}</div>
        <div className="col-2">
          <button
            type="button"
            className="btn btn-danger kg-button"
            onClick={() => onDeleteClick(todoName)}//grandparent ko return kiya
           >Delete </button>
        </div> </div> </div> ); } export default TodoItem;
```

WelcomeMessage.jsx

```
import styles from "./WelcomeMessage.module.css";
const WelcomeMessage = () => {
  return <p className={styles.welcome}>Enjoy Your Day</p>;
}; export default WelcomeMessage;
```

WelcomeMessage.module.css

```
.welcome { font-size: 30px; margin-top: 50px; font-weight: 600; }
```

*** We should be PRO till this concept at least.

28. React-icon Library

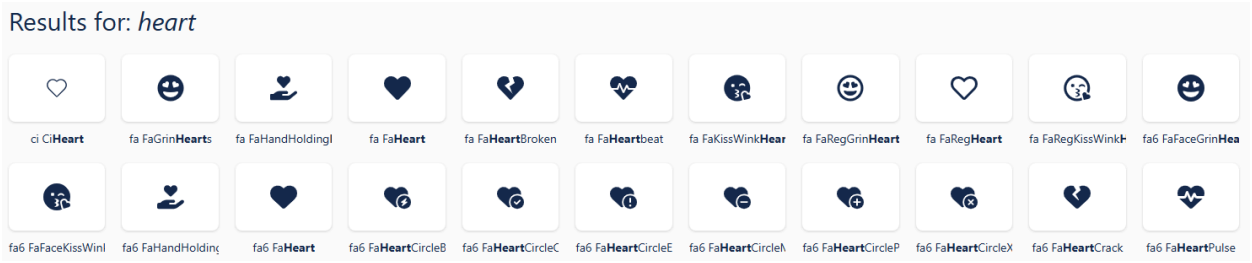
1. You can use a lot of icons without managing them.

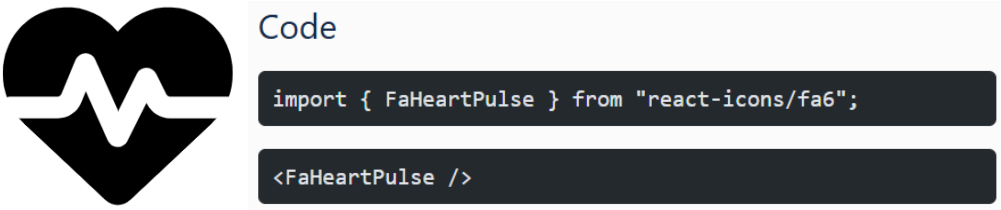
   URL: https://react-icons.github.io/react-icons/

2. Install Package npm install react-icons --save

3. Use icon: import { FaHeartPulse } from "react-icons/fa6";

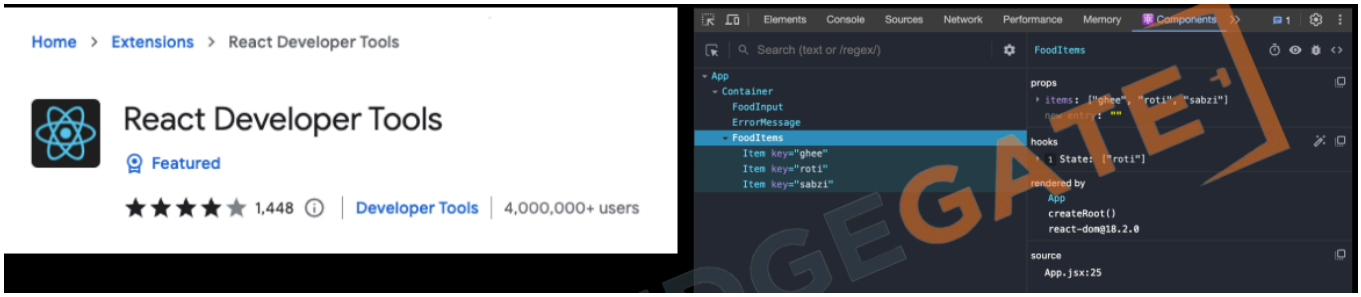
4. Use as a component: <FaHeartPulse></FaHeartPulse> Or <FaHeartPulse/>





29. Inspecting with React Dev Tools

1. Inspection: Allows inspection of React component hierarchies.
2. State & Props: View and edit the current state and props of components.
3. Performance: Analyze component re-renders and performance bottlenecks.
4. Navigation: Conveniently navigate through the entire component tree.
5. Filtering: Filter components by name or source to locate them quickly.
6. Real-time Feedback: See live changes as you modify state or props.



30. How React Works

Root Component:

• The App is the main or root component of a React application.

• It's the starting point of your React component tree.

Virtual DOM:

• React creates an in-memory structure called the virtual DOM.

• Different from the actual browser DOM.

• It's a lightweight representation where each node stands for a component and its attributes.

Reconciliation Process:

• When component data changes, React updates the virtual DOM's state to mirror these changes.

• React then compares the current and previous versions of the virtual DOM.

• It identifies the specific nodes that need updating.

• Only these nodes are updated in the real browser DOM, making it efficient.

React and ReactDOM:

• The actual updating of the browser's DOM isn't done by React itself.

• It's handled by a companion library called react-dom.

Root Element:

• The root div acts as a container for the React app.

• The script tag is where the React app starts executing.

• If you check main.tsx, the component tree is rendered inside this root element.

Strict Mode Component:

• It's a special component in React.

• Doesn't have a visual representation.

• Its purpose is to spot potential issues in your React app.

Platform Independence:

• React's design allows it to be platform-agnostic.

• While react-dom helps build web UIs using React, ReactNative can be used to craft mobile app UIs.

React: Elements se deal karta hai.

ReactDom: Virtual and RealDom se deal krta hai.

31. React Vs Angular vs Vue

React, Angular, and Vue:

• React is a library, while Angular and Vue.js are frameworks.

• React focuses on UI; Angular and Vue.js offer comprehensive tools for full app development.

Library vs. Framework:

• A library offers specific functionality.

• A framework provides a set of tools and guidelines.

• In simpler terms: React is a tool; Angular and Vue.js are toolsets.

React's Specialty:

• React's main role is crafting dynamic, interactive UIs.

• It doesn't handle routing, HTTP calls, state management, and more.

React's Flexibility:

• React doesn't dictate tool choices for other app aspects.

• Developers pick what fits their project best.

About Angular and Vue.js:

• Angular, developed by Google, provides a robust framework with a steep learning curve.

• Vue.js is known for its simplicity and ease of integration, making it beginner-friendly.

32. Using Forms (mainly tb use krte hain jb hme form ka data server pe bhejna ho**)**

1. State Management: Each input's state is stored in the component's state.

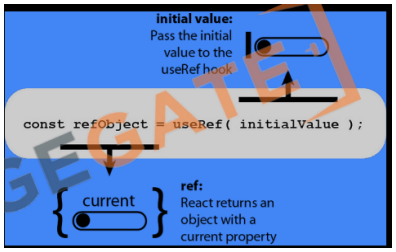
2. Handling Changes: Use onChange to detect input changes.

3. Submission: Utilize onSubmit for form submissions and prevent default with event.preventDefault(). //isse by-default behavior server pe save krna prevent ho jayega

4. Validation: Implement custom validation or use third-party libraries.

33. Use Ref

1. useRef allows access to DOM elements and retains mutable values without re-renders.
2. Used with the ref attribute for direct DOM interactions.
3. Can hold previous state or prop values.
4. Not limited to DOM references; can hold any value.
5. Refs can be passed as props also.



useState: isme state jb v change hoti hai to UI re-paint hoti hai.

useRef: isme state jb v change ho to state save krke rakhna hai pr UI re-paint nhi krna hai tb use kr skte hain.

AddToDo.jsx using useState hook

```
import { useState } from "react";
function AddTodo({ onNewItem }) {
  const [todoName, setTodoName] = useState("");
  const [dueDate, setDueDate] = useState("");
  const handleNameChange = (event) => {
    setTodoName(event.target.value); };
  const handleDateChange = (event) => {
    setDueDate(event.target.value); };
  const handleAddButtonClicked = () => {
    onNewItem(todoName, dueDate); //parent ko 2 chij return kr rha
    setDueDate(""); setTodoName(""); };
  return (
<div className="container text-center">
 <div className="row kg-row">
  <div className="col-6">
   <input type="text" placeholder="Enter Todo Here"
      value={todoName} onChange={handleNameChange} /> </div>
  <div className="col-4">
   <input type="date" value={dueDate} onChange={handleDateChange}/> </div>
  <div className="col-2">
    <button type="button" className="btn btn-success kg-button"
      onClick={handleAddButtonClicked}>Add </button>
       </div> </div> </div> );}export default AddTodo;
```

AddToDo.jsx using useRef hook

```
import {useRef } from "react"; //import kr liye
function AddTodo({ onNewItem }) {
  const todoNameElement = useRef(); //variable bnaye
  const dueDateElement = useRef(); //initial value v chaho to pass kr lo
  const handleAddButtonClicked = () => {
    const todoName = todoNameElement.current.value;
    const dueDate = dueDateElement.current.value;
    todoNameElement.current.value ="";
    dueDateElement.current.value ="";//baar baar re-render nhi hoga, jb chahiye value tb ka le lo
    onNewItem(todoName, dueDate); };
  return (
    <div className="container text-center">
      <div className="row kg-row">
        <div className="col-6">
          <input
            type="text"
            ref={todoNameElement} //variable use kr rhe
            placeholder="Enter Todo Here"
          />
        </div>
        <div className="col-4">
          <input type="date" ref={dueDateElement}/> //variable use kr rhe
        </div>
        <div className="col-2">
          <button type="button"
            className="btn btn-success kg-button"
            onClick={handleAddButtonClicked} > Add
          </button>
        </div>
      </div>
    </div>
  );
}export default AddTodo;
```

34. Update state from Previous State

• **Spread Operator**: Use to maintain immutability when updating arrays or objects.

• **Functional Updates**: Use (existingPosts) => [postData, ...existingPosts] to avoid stale values during asynchronous updates.

Spread Operator:

Var arr1 = [1,2,3];

Var arr2 = [arr1, 4, 5];

Print arr2 [o/p: arr1, 4, 5] total 3 elements only kyunki array ka koi type nhi hota.

Var arr3 = [...arr1, 4, 5] [... is known as spread operator]

Print arr3 [o/p: 1,2,3,4,5]

```
const [todoItems, setTodoItems] = useState([]);
const handleNewItem = (itemName, itemDueDate) => {
    const newTodoItems = [
      ...todoItems,
      { name: itemName, dueDate: itemDueDate },
    ];
    setTodoItems(newTodoItems);
  }; //kvi kvi ye galat/old value v return kr skta hai jb mera application bhot complicated to jaye toh..isliye we can use functional updates as an alternative
```

Functional Updates:

```
const [todoItems, setTodoItems] = useState([]);
const handleNewItem = (itemName, itemDueDate) => {
    setTodoItems((currValue)=>{
      const newTodoItems = [
        ...currValue,
        { name: itemName, dueDate: itemDueDate },
      ];
      return newTodoItems;
    });
  };
```

35. Context API (It's like a common shared storage among all the components)

1. Prop Drilling: Context API addresses prop drilling

2. Folder Setup: Use a store/context folder for context files.

Context API: A way to pass data through the component tree without having to pass props down manually at every level.

createContext: Creates a context object.

Provider: A component that provides the context value to its children.

useContext(consumer): A hook that allows you to consume a context.

App
|
Counter;
|
Child
|
GrandChild
|
GrandGrandChild

Prop Drilling: Agar hme koi property counter se GrandGrandChild ko pass krni hai to react me aisa method nhi h jisse directly pass kr ske one by one parent to child hi karna padta tha. Isi problem ko solve krne ke liye aaya Context API.

App.jsx

```
import './App.css'
import { Component1, Component2 } from './Components';
import Container from './Components/Container';
function App() {
 return (
    <>
      <Container>
          <Component1/> <Component2/>
      </Container>
    </> )} export default App;
```

ValueContext.jsx

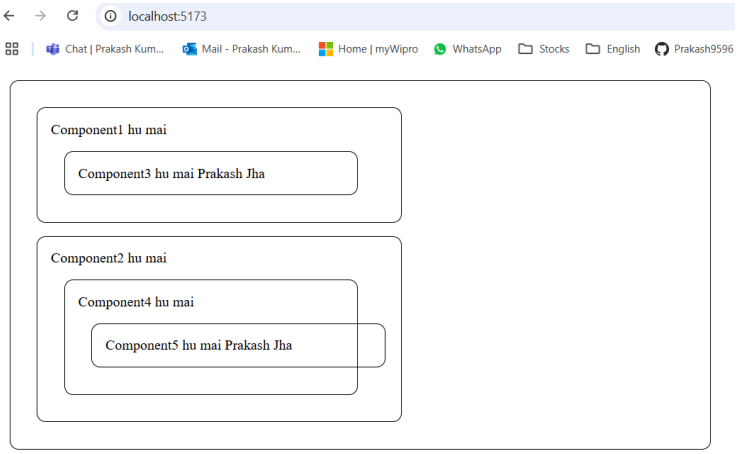
```
import { createContext } from "react";
const ValueContext = createContext('Prakash Jha');
export default ValueContext; //create krke initial value pass kiye
```

Component5.jsx

```
import { useContext } from "react";
import ValueContext from "../Context/ValueContext";
import Container from "./Container"
let Component5 = () => {
  const exactValue = useContext(ValueContext); //Comp5 me use kiye
  return ( <Container>
    <div>Component5 hu mai {exactValue}</div>
    </Container> ); }export default Component5;
```

Component3.jsx

```jsx
import { useContext } from "react";
import ValueContext from "../Context/ValueContext";
import Container from "./Container"
let Component3 = () => {
  const exactValue = useContext(ValueContext); //Comp3 me use kiye
  return ( <Container>
    <div>Component3 hu mai {exactValue}</div>
    </Container>
  ); }export default Component3;
```



Note: Pahle same chij ke liye prop drilling krna pdta ab bs ek context file bnaye aur jisko v use krna h jis bhi level pe wo direct usko use kr liya. Mainly ye tb use kr rhe jb same state bhot sare components me share krni hai.

36. Use Reducer

1. useReducer is a hook in React that offers more control over state operations compared to useState, especially for complex state logic. (State management Complex ho jane pe)
2. **Components**: It involves two main components:

Reducer: A pure function that takes the current state and an action and returns a new state.

Action: An object describing what happened, typically having a type property.

3. Initialization: It's invoked as const [state, dispatch] = useReducer(reducer, initialState).
4. Dispatch: Actions are dispatched using the dispatch function, which invokes the reducer with the current state and the given action.
5. Use Cases: **Particularly useful for managing state in large components or when the next state depends on the previous one.**
6. Predictable State Management: Due to its strict structure, it leads to more predictable and maintainable state management.

Project: ToDo App (version:3/Final) : Isme Context API use krte hai

App.jsx

```
import AppName from "./components/AppName";
import AddTodo from "./components/AddTodo";
import TodoItems from "./components/TodoItems";
import WelcomeMessage from "./components/WelcomeMessage";
import "./App.css";
import { useState } from "react";
import { TodoItemsContext } from "./store/todo-items-store";
function App() {
  const [todoItems, setTodoItems] = useState([]); //to set state
  const addNewItem= (itemName, itemDueDate) => {
  const newTodoItems = [
      ...todoItems,{ name: itemName, dueDate: itemDueDate },];//spread operator
    setTodoItems(newTodoItems);};
  const deleteItem= (todoItemName) => {
  const newTodoItems = todoItems.filter((item) => item.name !==
   todoItemName); //ye condition satisfy karne wala new array bnao
setTodoItems(newTodoItems); };
  return ( <TodoItemsContext.Provider
        value={{todoItems,addNewItem, deleteItem }}>
    <center className="todo-container">
      <AppName />
      <AddTodo />
      <WelcomeMessage />
      <TodoItems />
    </center>
    </TodoItemsContext.Provider> ); } export default App;
//C.API ka Provider ye provider jo v use karna chahega ye sab ko 1
variable aur 2 function de skta h [todoItems,addNewItem, deleteItem]
->Context.Provider basically wo hota hai jo apne apne aane wale sare
components ko wo service provide krta hai jo Context me available h
```

store/todo-items-store.jsx

```
import {createContext} from "react";
export const TodoItemsContext = createContext(
todoItems: [],
addNewItem: () => {},
deleteItem: () => {},);
```

App.css

```
input { width: 100%; }
.kg-button { min-width: 80px; }
.kg-row { margin: 10px 5px; }
```

AppName.jsx

```
import styles from "./AppName.module.css";
function AppName() {
  return <h1 className={styles.todoHeading}>TODO App</h1>;
}export default AppName;
```

AppName.module.css

```
.todoHeading { font-weight: 700; font-size: 45px;
  margin: 10px; margin-bottom: 20px; }
```

TodoItems.jsx **//C.API ko TodoItems use krna chahta hai**

```
import TodoItem from "./TodoItem";
import styles from "./TodoItems.module.css";
import { TodoItemsContext } from "./store/todo-items-store";
Import { useContext } from "react"; //Import stmt likhega
const TodoItems = () => {
const {todoItems } = useContext(TodoItemsContext);//useContext se value lega
  return (
    <div className={styles.itemsContainer}>
      {todoItems.map((item) => ( //use kr lega
        <TodoItem
          todoDate={item.dueDate}
          todoName={item.name}
        ></TodoItem>
      ))} </div> ); }; export default TodoItems;
```

TodoItems.module.css

```
.itemsContainer { text-align: left;}
```

WelcomeMessage.jsx **//C.API ko WelcomeMessage use krna chahta hai**

```
import styles from "./WelcomeMessage.module.css";
import { TodoItemsContext } from "./store/todo-items-store";
Import { useContext } from "react"; //Import stmt likhega
const WelcomeMessage = () => {
  const {todoItems} = useContext(TodoItemsContext);//useContext se value lega
  return todoItems.length ===0 && <p className={styles.welcome}>Enjoy Your
Day</p>; //use kr lega
}; export default WelcomeMessage;
```

AddToDo.jsx //C.API ko AddToDo use krna chahta hai

```
import { useState } from "react";
import { TodoItemsContext } from "./store/todo-items-store";
Import { useContext } from "react"; //Import stmt likhega
function AddTodo() {
  const {addNewItem} = useContext(TodoItemsContext); //useContext se value lega
  const [todoName, setTodoName] = useState("");
  const [dueDate, setDueDate] = useState("");
  const handleNameChange = (event) => {
    setTodoName(event.target.value); };
  const handleDateChange = (event) => {
    setDueDate(event.target.value); };
  const handleAddButtonClicked = () => {
    addNewItem(todoName, dueDate); //use kr lega
    setDueDate(""); setTodoName(""); };
  return (
    <div className="container text-center">
      <div className="row kg-row">
        <div className="col-6">
          <input
            type="text"
            placeholder="Enter Todo Here"
            value={todoName}
            onChange={handleNameChange}
          />
        </div>
        <div className="col-4">
          <input type="date" value={dueDate} onChange={handleDateChange}/>
        </div>
        <div className="col-2">
          <button
            type="button"
            className="btn btn-success kg-button"
            onClick={handleAddButtonClicked}>Add
          </button>
        </div>
      </div>
    </div>
  );}export default AddTodo;
```

WelcomeMessage.module.css

```
.welcome { font-size: 30px; margin-top: 50px; font-weight: 600; }
```

TodoItem.jsx **//C.API ko TodoItem use krna chahta hai**

```
import { TodoItemsContext } from "./store/todo-items-store";
Import { useContext } from "react"; //Import stmt likhega
function TodoItem({ todoName, todoDate}) {
const {deleteItem } = useContext(TodoItemsContext); //useContext se value lega
  return (
    <div className="container">
      <div className="row kg-row">
        <div className="col-6">{todoName}</div>
        <div className="col-4">{todoDate}</div>
        <div className="col-2">
          <button
            type="button"
            className="btn btn-danger kg-button"
            onClick={() => deleteItem(todoName)} //use kr lega
           >Delete </button>
        </div> </div> </div> ); } export default TodoItem;
```

Project: ToDo App (version:3/Final) : Isme useReducer use karte hai

App.jsx

```
import AppName from "./components/AppName";
import AddTodo from "./components/AddTodo";
import TodoItems from "./components/TodoItems";
import WelcomeMessage from "./components/WelcomeMessage";
import "./App.css";
import TodoItemsContextProvider from "./store/todo-items-store";
function App() {
 return (<TodoItemsContextProvider> //ye component context file me bnaye h
    <center className="todo-container">
      <AppName />
      <AddTodo />
      <WelcomeMessage />
      <TodoItems />
    </center>  </TodoItemsContextProvider>
    ); } export default App;
```

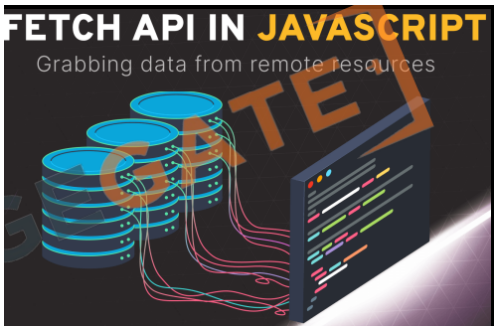
store/todo-items-store.jsx

```
import {createContext} from "react";
import { useReducer} from "react"; //reducer ko import kr liye
export const TodoItemsContext = createContext(
todoItems: [],
addNewItem: () => {},
deleteItem: () => {},);

Const TodoItemsReducer = (currTodoItems, action) => {
  let newTodoItems = currTodoItems; //currTodoItems todoItems hua
  if(action.type = = = 'NEW_ITEM'){
     newTodoItems = [
      ...currTodoItems,
  {name: action.payload.itemName, dueDate: action.payload.itemDueDate },];
   }else if (action.type = = = 'DELETE_ITEM'{
      newTodoItems = currTodoItems.filter((item) => item.name !==
   action.payload.todoItemName);
    }
 return newTodoItems; //newTodoItems dispatchTodoItems hua
};// ye ek aisa pure fun h jisko current state aur action do ye new state dega

const TodoItemsContextProvider = ({children}) => {
  const [ todoItems, dispatchTodoItems] = useReducer(todoItemsReducer, []);
//Ye useReducer 2 chij return kr rha h aur as an input  ek pure function aur ek initial value leta hai.
  const addNewItem= (itemName, itemDueDate) => {
   const newItemAction = {
    type: "NEW_ITEM", payload: {itemName, itemDueDate},
   }; dispatchTodoItems(newItemAction);
   }; //ye TodoItemsReducer ko call krke action pass kr rha h

 const deleteItem= (todoItemName) => {
  const deleteItemAction = {
   type: "DELETE_ITEM", payload: {todoItemName,},
   }; dispatchTodoItems(deleteItemAction );
}; //ye bhi TodoItemsReducer ko call krke action pass kr rha h

 return <TodoItemsContext.Provider
        value={{todoItems,addNewItem, deleteItem }}> {children}
<TodoItemsContext.Provider>
}export default TodoItemsContextProvider;
```

Project : Social Media APP

38. Data fetching using Fetch

1. fetch: Modern JavaScript API for network requests.
2. Promise-Based: Returns a Promise with a Response object.
3. Usage: Default is GET. For POST use method: 'POST'
4. Response: Use .then() and response.json() for JSON data.
5. Errors: Doesn't reject on HTTP errors. Check response.ok.
6. Headers: Managed using the Headers API.




```
fetch('https://dummyjson.com/posts')
.then(res => res.json())
.then(obj => console.log(obj.posts));
```

Social media APP me fetch API ka use karke data server se la rhe

post-list-store.jsx

```
export const PostList = createContext({
  postList: [],
  addPost: () => {},
  addInitialPosts: () => {},
  deletePost: () => {},
});
const postListReducer = (currPostList, action) => {
  let newPostList = currPostList;
  if (action.type === "DELETE_POST") {
    newPostList = currPostList.filter(
      (post) => post.id !== action.payload.postId );
  } else if (action.type === "ADD_INITIAL_POSTS") {
    newPostList = action.payload.posts;
  } else if (action.type === "ADD_POST") {
    newPostList = [action.payload, ...currPostList];
  } return newPostList; };
```

```
const addInitialPosts = (posts) => {
    dispatchPostList({
      type: "ADD_INITIAL_POSTS",
      payload: {
        posts,
      },
    });
  };
<PostList.Provider value={{ postList, addPost, addInitialPosts,
deletePost}}>
      {children}
    </PostList.Provider>
```

WelcomeMessage.jsx

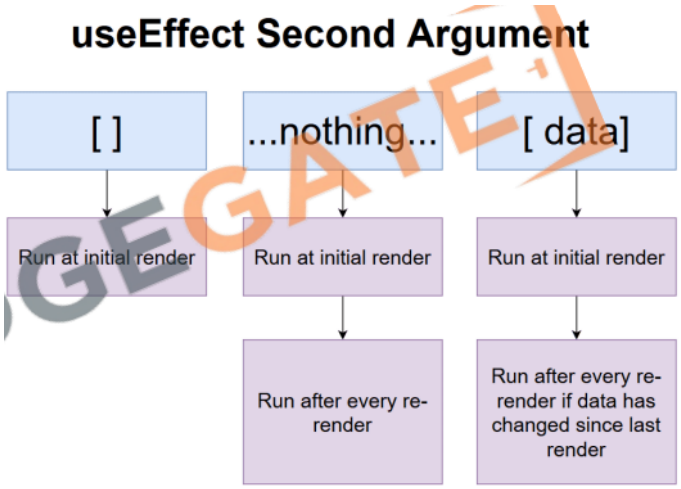
```
const WelcomeMessage = ({onGetPostsClick}) => {
  return <center className="welcome-message">
    <h1>There are no Posts</h1>
    <button type="button" onClick={onGetPostsClick} className="btn
btn-primary">Get Posts From Server</button>
  </center>
}
```

PostList.jsx

```
export default WelcomeMessage;
import { useContext } from "react";
import Post from "./Post";
import { PostList as PostListData } from "../store/post-list-store";
import WelcomeMessage from "./WelcomeMessage";
const PostList = () => {
  const { postList, addInitialPosts } = useContext(PostListData);
  const handleGetPostsClick = () => {
    fetch('https://dummyjson.com/posts')
    .then(res => res.json())
    .then(data => {addInitialPosts(data.posts);}); };
  return (
    <> { postList.length === 0 && <WelcomeMessage
onGetPostsClick={handleGetPostsClick}/> }
      {postList.map((post) => (
        <Post key={post.id} post={post} />
      ))}
    </> ); }; export default PostList;
```

39. The useEffect Hook (automatically data server se fetch hoga page render hone pe, kuchh condition bhi daalna chaho to dal lo **)**

1. In function-based components, useEffect handles side effects like data fetching or event listeners.
2. useEffect runs automatically after every render by default.
3. By providing a dependency array, useEffect will only run when specified variables change. An empty array means the effect runs once.
4. Multiple useEffect hooks can be used in a single component for organizing different side effects separately.



useEffect me hm 2 argument dete hain, ek to function.

2nd argument: agar [] de rhe to ye function bs first time page load hone pe hi chlega.

Agar kuchh v nhi de rhe, to ye hr state ke change hone pe run kr dega.

Agar kuchh pass kr rhe jaise koi prop ya state ya koi specific condition to wo ush particular condition pe hi ush fun ko run krega + initially first time page load hone pe.

Social media APP me useEffect hook ka use

WelcomeMessage.jsx

```
const WelcomeMessage = () => {
  return (
    <center className="welcome-message">
      <h1>There are no posts</h1>
    </center>
  );
};
export default WelcomeMessage;
```

Isse Button hta diye, jisko click krke pahle data fetch krte the

PostList.jsx

```
export default WelcomeMessage;
import { useContext } from "react";
import Post from "./Post";
import { PostList as PostListData } from "../store/post-list-store";
import WelcomeMessage from "./WelcomeMessage";
const PostList = () => {
  const { postList, addInitialPosts } = useContext(PostListData);
  useEffect(() => {
  fetch('https://dummyjson.com/posts')
    .then((res) => res.json())
    .then((data) => {addInitialPosts(data.posts);});
},[]) //useEffect me 2 argument de rhe {function aur []}
        return (
    <> { postList.length === 0 && <WelcomeMessage/> }
      {postList.map((post) => (
        <Post key={post.id} post={post} />
      ))}
    </> ); }; export default PostList;
```

40. Handling Loading State (taki jbtk load ho rha data server se tbtk ye dikhe)

PostList.jsx

```
export default WelcomeMessage;
import { useContext } from "react";
import Post from "./Post";
import { PostList as PostListData } from "../store/post-list-store";
import WelcomeMessage from "./WelcomeMessage";
const PostList = () => {
  const { postList, addInitialPosts } = useContext(PostListData);
  const [fetching, setFetching] = useState(false); //ek state bnaye
  useEffect(() => {
  setFetching(true); //jaise hi fetching start ho ye true ho jaye
  fetch('https://dummyjson.com/posts')
```

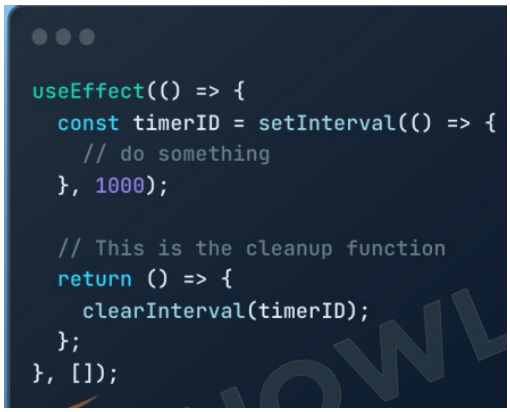
```
    .then((res) => res.json())
    .then((data) => {addInitialPosts(data.posts);
     setFetching(false); }); },[])
        return ( //jaise hi fetching end ho ye false ho jaye
    <> { fetching && <loadingSpinner/>} //jb true hai to spinner dikhao
       { !fetching && postList.length === 0 && <WelcomeMessage/> }
      {!fetching && postList.map((post) => (
        <Post key={post.id} post={post} />
      ))} //jb welcomeMessage ya postList hai to spinner mt dikhao
    </> ); }; export default PostList;
```

LoadingSpinner.jsx

```
const LoadingSpinner = () => {
return <h1> Loading...Bootstrap ko tip ke koi spinner daal do idhar</h1>
}
export default LoadingSpinner; //ek component bna liye
```

41. The useEffect Hook Cleanup

Returning a function from `useEffect` allows for cleanup, ideal for removing event listeners.

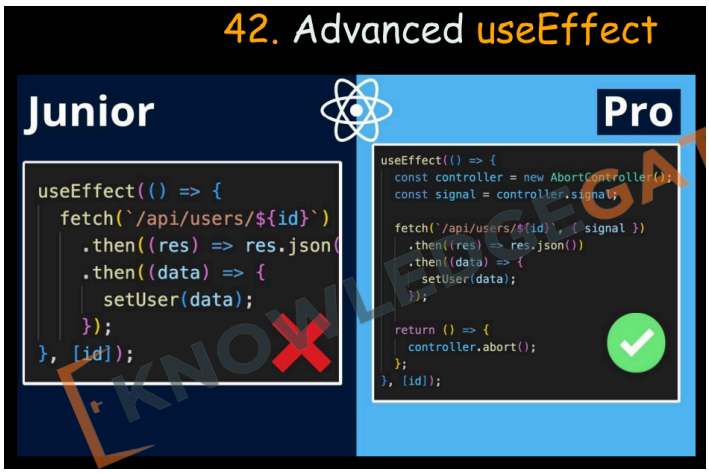


```
useEffect(() => {
  const timerID = setInterval(() => {
    // do something
  }, 1000);

  // This is the cleanup function
  return () => {
    clearInterval(timerID);
  };
}, []);
```

Ye important tb hota hai maan lijiye aap server se koi data fetch kr rhe hain pr bich me hi aap page badal kar kahin aur chale gaye maan lijiye ab aapka wo dekhne ka mood nhi h, to wo background me fetch karta hi rhega jisse ye app slow v ho jayega aur faltu me user ka data v waste hoga ..isliye useEffect ke return se hm ek method call karwa denge jiska use tb ho jayega aur wo fetching stop kr dega.

PostList.jsx

```
export default WelcomeMessage;
import { useContext } from "react";
import Post from "./Post";
import { PostList as PostListData } from "../store/post-list-store";
import WelcomeMessage from "./WelcomeMessage";
const PostList = () => {
  const { postList, addInitialPosts } = useContext(PostListData);
  const [fetching, setFetching] = useState(false);
  useEffect(() => {
  setFetching(true);
  const controller = new AbortController();
  const signal = controller.signal;
  fetch('https://dummyjson.com/posts', {signal})
    .then((res) => res.json())
    .then((data) => {addInitialPosts(data.posts);
     setFetching(false); });
     return () => {
     console.log ("page switch kar rahe ho, fetching stop kr rha hu")
     controller.abort();
   } //abort maarne wala logic likh diye if needed
     },[]);
       return ( //jaise hi fetching end ho ye false ho jaye
    <> { fetching && <loadingSpinner/>} //jb true hai to spinner dikhao
       { !fetching && postList.length === 0 && <WelcomeMessage/> }
      {!fetching && postList.map((post) => (
        <Post key={post.id} post={post} />
      ))} //jb welcomeMessage ya postList hai to spinner mt dikhao
    </> ); }; export default PostList;
```



Project: Bharat-clock-version-2

App.jsx

```
import ClockHeading from "./components/ClockHeading";
import ClockSlogan from "./components/ClockSlogan";
import CurrentTime from "./components/CurrentTime";
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";
function App() {
  return (
    <center>
      <ClockHeading></ClockHeading>
      <ClockSlogan></ClockSlogan>
      <CurrentTime></CurrentTime>
    </center> ); } export default App;
```

ClockHeading.jsx

```
let ClockHeading = () => {
  return <h1 className="fw-bolder">Bharat Clock</h1>;
}; export default ClockHeading;
```

ClockSlogan.jsx

```
let ClockSlogan = () => {
  return (
    <p className="lead">
      This is the clock that shows the time in Bharat at all times.
    </p> ); }; export default ClockSlogan;
```

CurrentTime.jsx

```
import { useEffect, useState } from "react";
let CurrentTime = () => {
  const [time, setTime] = useState(new Date());
  useEffect(() => {
    const intervalId = setInterval(() => {
      setTime(new Date());
    }, 1000); //hr ek sec ke baad setTime ko update kr dena
    return () => {
      clearInterval(intervalId); }; }, []);
  return ( //component change krne pe setTime update karna bnd krega based on intervalId
    <p className="lead">
      This is the current time: {time.toLocaleDateString()} -{" "}
      {time.toLocaleTimeString()} </p>
  ); }; export default CurrentTime;
```

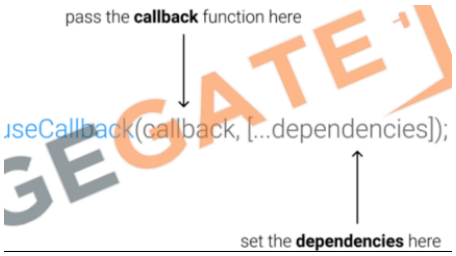
App do baar render kyu hota hai ?

Reason : <React.StrictMode/>

Solution: Real production me nhi hoga aisa kyunki <React.StrictMode/> shift development ke waqt use hota hai.

43. The useCallback Hook

1. Memoization: Preserves function across renders to prevent unnecessary re-renders.
2. Optimization: Enhances performance in components with frequent updates.
3. Dependency Array: Recreates the function only when specific dependencies change.
4. Event Handlers: Used to keep consistent function references for child components.
5. With useEffect: Prevents infinite loops by maintaining function references.



useEffect jaisa hi hai.

Fayda: Isse aisa kr skte ki Parent re-paint hoga tb bhi jaruri nhi hai ki child repaint ho hi.

Ex: const add = (num1, num2) => num1 + num2;

const add2=(num1, num2) => num1 + num2;

add === add2 (false)

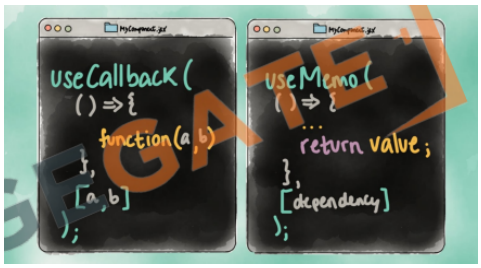
Mtlb definition same hai tb v agar function dubara define hota h to wo naye object ke tarah treat hota hai.

Mtlb maan lijiye deletePost() methd hai jo jitne baar call hoga utne baar ye method naye object ke tarah treat hoga aur condition basis pe page re-paint hoga: ye to thk hai ..pr ye addItems() agar kisi pe dependent hai to iske wajah se wo v re-paint hoga, because of its dependency: ye bhot galat baat ka isi ka solution hai useCallback

```
const deletePost = useCallback((postId) => {
    dispatchPostList({
      type: "DELETE_POST",
      payload: {
        postId,
      },
    });
  }, [dispatchPostList]); //jb dispatchPostList change ho tvi ish method ka naya object create ho bs, baad baki kisi case me na ho.
```

44 The useMemo Hook

1. Memoization: useMemo caches the result of expensive calculations to enhance performance.
2. Re-computation: Only re-computes the memoized value when specific dependencies change.
3. Optimization: Helps prevent unnecessary recalculations, improving component rendering efficiency.
4. Dependency Array: Uses an array of dependencies to determine when to recompute the cached value.
5. Comparison with useCallback: While useCallback memoizes functions, useMemo memoizes values or results of functions.
6. Best Use: Ideal for intensive computations or operations that shouldn't run on every render.



Example: Maan lijiye mere paas 1000 students ka data aaya pahle hm usko sort kiye tb use kiye. Ab hm to chahenge nhi ki ye kaam hr baar ho jb required na v ho isliye aise value/data/data-structure ko tvi change karenge jb required ho, nhi to cache memory me ye useMomo rakha rakhega baar baar same chij nhi krega.

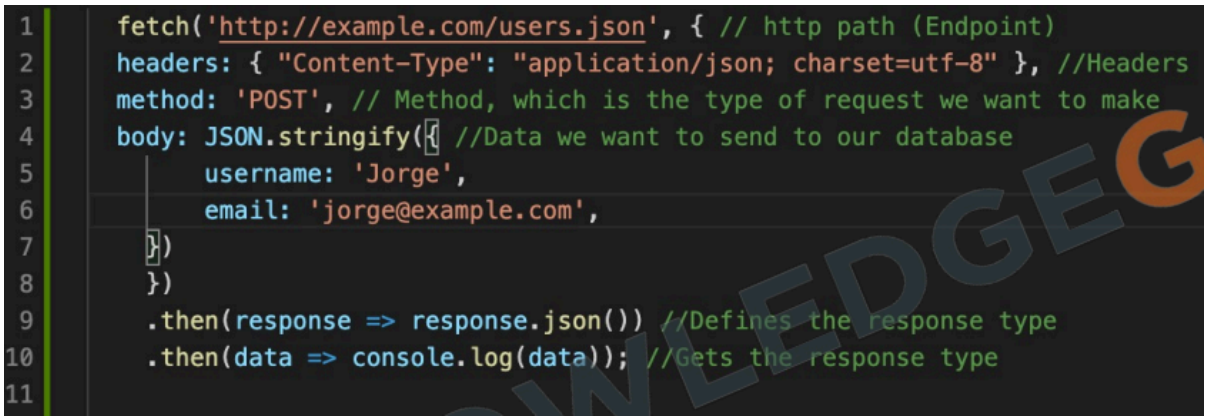
```
const arr = [5,2,6,7,4];
const sortedArr = useMemo( () => arr.sort(), (arr)); //tvi sort hoga jb array change hoga
```

useCallback Function preserve krta hai aur useMemo value preserve krta hai for better performance.

45. Custom Hooks

1. Reusable Logic: Custom hooks allow you to extract and reuse component logic.
2. Naming Convention: Typically start with "use" (e.g., useWindowSize, useFetch).
3. Combining Hooks: Custom hooks can combine multiple built-in hooks
4. Sharing State: Enables sharing of stateful logic without changing component hierarchy.
5. Isolation: Helps in isolating complex logic, making components cleaner and easier to maintain.
6. Custom Return Values: Can return any value (arrays, objects, or any other data type) based on requirements.

46. Submitting data with Fetch



```
fetch('http://example.com/users.json', { // http path (Endpoint)
headers: { "Content-Type": "application/json; charset=utf-8" }, //Headers
method: 'POST', // Method, which is the type of request we want to make
body: JSON.stringify({ //Data we want to send to our database
    username: 'Jorge',
    email: 'jorge@example.com',
})
})
.then(response => response.json()) //Defines the response type
.then(data => console.log(data)); //Gets the response type
```

Avi tak data ab hm GET kiye hain server se, ish baar data POST krenge server pe.

Social Media app ke andar:

```
fetch("https://dummyjson.com/posts/add", {
   method: "POST",
   headers: { "Content-Type": "application/json" },
   body: JSON.stringify({
    title: postTitle,
    body: postBody,
    reactions: reactions,
    userId: userId,
    tags: tags,
 }),
})
 .then ((res) => res.json())
 .then(post => addPost(post)); //receive v ab karega wo ek value that is post itself.
```

//Ab jb hmlog web wala form bharenge to ye data server pe jayega aur phir wahan se aayega bhi .then & .then jo kr rhe ush line se. Aur same usi post ko addPost ko call krwa ke add kr de rhe.

47 React Router

1. Installation: Use **npm install react-router-dom**.
2. We are going to use the latest version which is 6+
3. RouterProvider: Wraps the app for routing capabilities.
4. createBrowserRouter: helps create the mapping for the router provider.
5. Declarative Routing: Easily define application routes.
6. Routes are React components.

Kisi bhi tarah ka URL routing karna hai to hum log React Router use krte hain.

Mera URL: localhost:5173/aage_kuchh_bhi_likho (hr baar homepage hi aayega)

Social Media APP

Main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { RouterProvider, createBrowserRouter } from "react-router-dom";
import App from "./routes/App.jsx";
import CreatePost from "./components/CreatePost.jsx";
import PostList from "./components/PostList.jsx";
const router = createBrowserRouter([
  {
    path: "/",                          /pe
    element: <App />,                   App dikhao aur uske child me se
    children: [
      { path: "/", element: <PostList />,},    agar / hai to postlist dikhao
      {path: "/create-post", element: <CreatePost />, },  agar/c-post hai to c-post dikhao
    ],
  },
]);
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router} /> //yahan pahle app hota tha
  </React.StrictMode> );
```

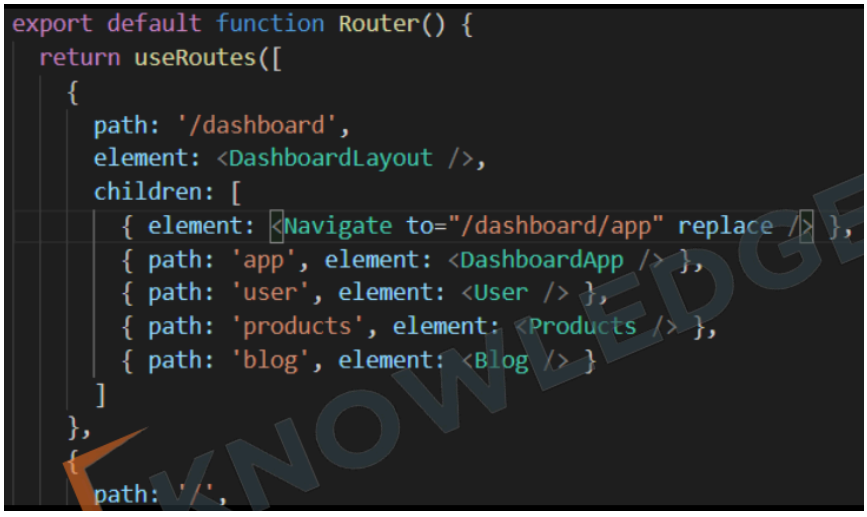
Note: App.jsx & App.css ko rotes folder me move kiye, simply taki routing wala sara chij ek jagah rhe

App.jsx

```
import { Outlet } from "react-router-dom"; //import v kr liye
function App() {
  const [selectedTab, setSelectedTab] = useState("Home");
  return (
    <PostListProvider>
      <div className="app-container">
        <Sidebar
          selectedTab={selectedTab}
          setSelectedTab={setSelectedTab}
        ></Sidebar>
        <div className="content">
          <Header></Header>
          <Outlet />//ye component hmlog bnaye nhi hain ye react ka h apna
          <Footer></Footer>
        </div>
      </div>
    </PostListProvider>
  );
}
```

48. Layout Routes

1. Layout Routes help us to use shared elements (jo tree jaisa struct. Bnaye hain)
2. Outlet component is used to render the children at the correct places (Main.jsx wala)



SideBar.jsx

```
<a href="/" className="nav-link text-white" aria-current="page">
   <svg className="bi pe-none me-2" width="16" height="16">
        <use xlinkHref="#home"></use>
    </svg> Home </a>

<a href="/create-post" className="nav-link text-white">
     <svg className="bi pe-none me-2" width="16" height="16">
          <use xlinkHref="#speedometer2"></use>
      </svg> Create Post </a>
```

Note: Isse routing to ho gya, lekin ye Single page application nhi rha

Solution:

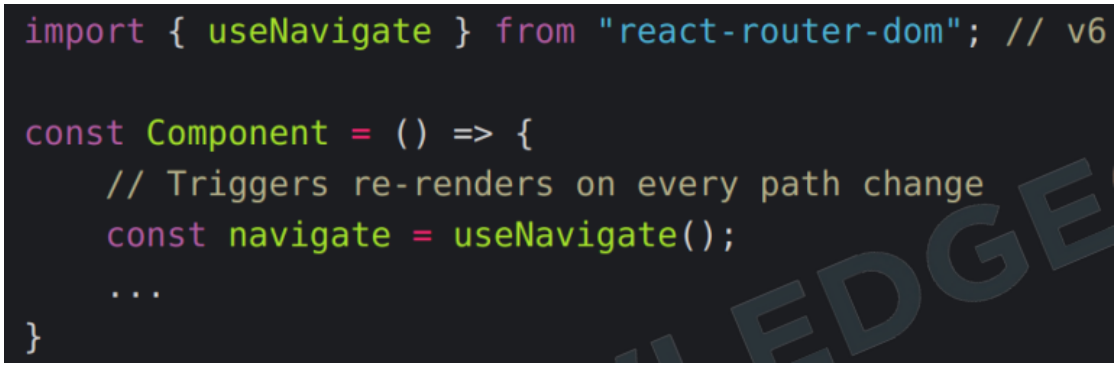
SideBar.jsx

```
<Link to="/" className="nav-link text-white" aria-current="page">
   <svg className="bi pe-none me-2" width="16" height="16">
        <use xlinkHref="#home"></use>
    </svg> Home </Link>

<Link to="/create-post" className="nav-link text-white">
     <svg className="bi pe-none me-2" width="16" height="16">
          <use xlinkHref="#speedometer2"></use>
      </svg> Create Post </Link>
```

Isse routing v ho gya aur SPA ka fayda v utha liye

49. Route Links (Dynamic routing karenge: useNavigate hook ka use karke)

1. Link Component with to property can be used to avoid reloading
2. useNavigate hook can be used to do navigation programmatically.



Mann lijiye create post wala details daale aur post pe click kiye to wo automatically all post wale pe chale jana chahiye na ..uske liye

CreatePost.jsx

```
import {useNavigate} from "react-router-dom";
const CreatePost = () => {
  const navigate = useNavigate();
....
fetch("https://dummyjson.com/posts/add", {
   method: "POST",
   headers: { "Content-Type": "application/json" },
   body: JSON.stringify({
    title: postTitle,
    body: postBody,
    reactions: reactions,
    userId: userId,
    tags: tags,
 }),
})
 .then ((res) => res.json())
 .then(post => addPost(post)
 navigate("/"); );  //Add post hone ke just baad ye automatically homepage/Post page pe aa jayega
```

50. Data fetching using loader

1. Loader method can be used to load data before a particular route is executed.
2. The loader method must return the data that is loaded or promise.
3. Data is available in components and all the child components.
4. useLoaderData hook can be used to get the fetched data.
5. Loading state can also be used.

PostList.jsx

```
import { useLoaderData } from "react-router-dom";
const PostList = () => {
  const postList = useLoaderData();
  return ( //PostList render hone se pahle postLoader, hook ka use krke postList dega
    <>
      {postList.length === 0 && <WelcomeMessage />}
      {postList.map((post) => (
        <Post key={post.id} post={post} />
      ))} </>  ); };
export const postLoader = () => {
  return fetch("https://dummyjson.com/posts")
    .then((res) => res.json())
    .then((data) => {
      return data.posts;
    });
};ek function bnaye postLoader
```

Main.jsx

```
import { postLoader } from "./components/PostList.jsx";
{
    path: "/",
    element: <App />,
    children: [
      { path: "/", element: <PostList />, loader: postLoader },
      { path: "/create-post", element: <CreatePost />, },
    ],
  },
```

Note: Ish postLoader ka use ye h ki ye PostList ko render krne se pahle apna ye function execute krega tvi PostList ko render krega.

51. Submitting data using action

1. Action method can be used to perform an action on submission of Forms.
2. Custom Form component need to be used along with name attribute for all inputs.
3. Action function will get a data object. To generate correct request object method="post" attribute should be used.
4. Data.request.formData() method can be used to get form data Object.
5. Object.fromEntries(formData) can be used to get actual input data.
6. redirect() response can be returned for navigation after submission.

Main.jsx

```
import {createPostAction} from "./components/CreatePost.jsx";
{   path: "/",
    element: <App />,
    children: [
      { path: "/", element: <PostList />, loader: postLoader },
      { path: "/create-post", element: <CreatePost />, action:createPostAction},
    ], },
```

CreatePost.jsx

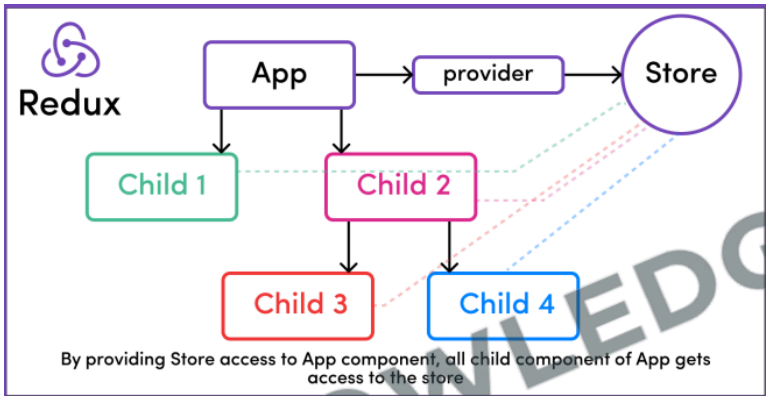
```
import { Form, redirect } from "react-router-dom";
const CreatePost = () => {
  return (
    <Form method="POST" className="create-post">
     ...
    <input type="text" name="userId" .../> </div>
     ...
    <input type="text" name="title" .../></div>
     ...
    <textarea type="text" name="body".../></div>
     ...
   <input type="text" name="reactions" .../> </div>
    ...
    <input type="text" name="tags".../></div>
      <button type="submit" className="btn btn-primary"> Post
      </button>
    </Form> ); }; //iske use se bhot sara code km hua
export async function createPostAction(data) {
  const formData = await data.request.formData();
  const postData = Object.fromEntries(formData);
  postData.tags = postData.tags.split(" ");
  console.log(postData);
  fetch("https://dummyjson.com/posts/add", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(postData),
  }).then((res) => res.json())
    .then((post) => {
      console.log(post);
    }); return redirect("/");
}export default CreatePost;
```

52. What is Redux

1. State management for cross component or app-wide state.
2. Redux is a predictable state management library for JavaScript apps.
3. Local State vs Cross-component state vs App-Wide state
4. useState or useReducer vs useState with prop drilling vs useState or useContext or Redux

Local State: Jo state shift ek component me use ho
Cross-Component: Jo multiple components ke liye ho
App-Wide state: Jo pure app ke liye ho

useContext ek application ke liye dher sare bnaye ja skte hain pr Redux pure application ke liye shirf ek banta hai.

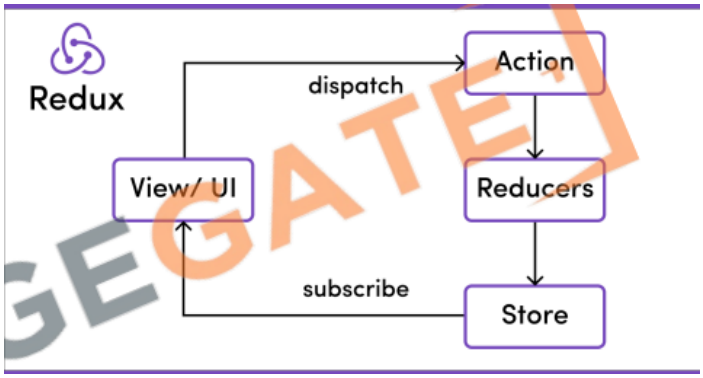


53. React-Context vs Redux

1. You can use both.
2. Setup and Coding is tough especially if you have multiple context providers.
3. Performance is slow. Context should only be used for things that rarely change. On the other hand Redux has great performance.
4. If these things don't matter to you then you can choose not to use redux and stay with React-Context.

54. How Redux Works

1. Single Source: Uses a single central store to maintain the entire application's state.
2. Actions: Components never directly change the store. Changes to state are made through dispatched actions, which describe events.
3. Reducers: Actions are processed by reducers, pure functions that return the new state.
4. Immutable: State is immutable; every change results in a new state object.
5. This is different from the useReducer hook.



Redux ke store se kuchh bhi chij chahiye to pahle subscribe kriye mtlb kuchh btaiye ki aap kis section ko use karna chah rhe kyunki store me to bhot kuchh hai n.
Agar aapko store me kuchh change karna hai to aap directly change nhi kr skte aap pahle dispatch krenge ek action aur ush action pe reducer act krega previous aur new state ko compare krke aur tb jake store me change krega.

55. Working with Redux

1. npm init –y
2. npm install redux
3. import in node: **const redux = require('redux');**
4. We need to setup all 4 basic things:
    1. Reducer
    2. Store
    3. Subscriber
    4. Actions
5. **node redux-demo.js** command to run node server

Fresh Node ke project me Redux ka use:

```
const redux = require('redux'); //redux import kiye
const INITIAL_VALUE = { counter: 0 }; //Initial value set kiye
const reducer = (store = INITIAL_VALUE, action) => {
  let newStore = store;
  if (action.type === 'INCREMENT') {
    newStore = {counter: store.counter + 1};
  } else if (action.type === 'DECREMENT') {
    newStore = {counter: store.counter - 1};
  } else if (action.type === 'ADDITION') {
    newStore = {counter: store.counter + action.payload.number};
  }return newStore; }//reducer fun bnaye jo action ke basis pe act krega
```

```
const store = redux.createStore(reducer); //store bnaye
const subscriber = () => {
  const state = store.getState();
  console.log(state);
} //subscriber function bnaye
store.subscribe(subscriber);
//subscriber ko call karke har baar state ka value dekhe
store.dispatch({type: 'INCREMENT'});
store.dispatch({type: 'DECREMENT'});
store.dispatch({type: 'INCREMENT'});
store.dispatch({type: 'ADDITION', payload: {number: 7}});
store.dispatch({type: 'DECREMENT'});
//Baar baar dispatcher ko call karke action pass kiye
```

Output:

```
{ counter: 1}
{ counter: 0}
{ counter: 1}
{ counter: 8}
{ counter: 7}
```

56. React with Redux

1. npm install redux
2. npm install react-redux
3. Create store folder with Index.js file
4. Creating the store using: import {createStore} from redux.
5. Providing the store with react
   1. Provider from react-redux
   2. <Provider store={store}><App/></Provider>
6. Using the store
   1. useSelector hook gets a slice of the store.
      Const counter = useSelector(state => state.counter);
   2. Subscription is already setup and only will re-execute when
      only your slice is changed. Subscription is automatically cleared also.
7. Dispatch Actions using the useDispatch hook.

Counter App

App.jsx

```
import "./App.css";
import "bootstrap/dist/css/bootstrap.min.css";
import Header from "./components/Header";
import DisplayCounter from "./components/DisplayCounter";
import Container from "./components/Container";
import Controls from "./components/Controls";
import { useSelector } from "react-redux";
import PrivacyMessage from "./components/PrivacyMessage";
function App() {
  const privacy = useSelector((store) => store.privacy);
  return ( //useSelector ka use krke store ka slice le rhe
    <center className="px-4 py-5 my-5 text-center">
      <Container>
        <Header></Header>
        <div className="col-lg-6 mx-auto">
          {privacy ? <PrivacyMessage /> : <DisplayCounter />}
          <Controls></Controls>
        </div> </Container> </center> ); } export default App;
```

Container.jsx

```
const Container = ({ children }) => {
  return (
    <div className="card" style={{ width: "70%" }}>
      <div className="card-body">{children}</div>
    </div> ); }; export default Container;
```

Header.jsx

```
const Header = () => {
  return <h1 className="display-5 fw-bold
text-body-emphasis">Counter</h1>;
};export default Header;
```

PrivacyMessage.jsx

```
const PrivacyMessage = () => {
  return <p className="lead mb-4">Counter is Private !!!!!</p>;
}; export default PrivacyMessage;
```

App.css

```
.control-row { margin-top: 10px; }
.number-input { max-width: 110px; }
```

DisplayCounter.jsx

```
import { useSelector } from "react-redux";
const DisplayCounter =() => {//store ka slice use kr rhe using useSelector
  const counter = useSelector((store) => store.counter);
  return <p className="lead mb-4">Counter current Value: {counter}</p>;
}; export default DisplayCounter;
```

Controls.jsx

```
import { useRef } from "react";
import { useDispatch } from "react-redux"; //useDispatch import kr liye
const Controls = () => {
  const dispatch = useDispatch(); //useDispatch function call kr rhe
  const inputElement = useRef();
  const handleIncrement = () => {
    dispatch({ type: "INCREMENT" }); };
  const handleDecrement = () => {
    dispatch({ type: "DECREMENT" }); };
  const handlePrivacyToggle = () => {
    dispatch({ type: "PRIVACY_TOGGLE" }); };
  const handleAdd = () => {
    dispatch({
      type: "ADD",
      payload: {
        num: inputElement.current.value,
      },
    });
    inputElement.current.value = "";
  };
  const handleSubtract = () => {
    dispatch({
      type: "SUBTRACT",
      payload: {
        num: inputElement.current.value,
      },
    });
    inputElement.current.value = "";
  };
```

```
return ( <>
      <div className="d-grid gap-2 d-sm-flex justify-content-sm-center">
        <button type="button" className="btn btn-primary"
          onClick={handleIncrement} >+1 </button>
        <button type="button" className="btn btn-success"
          onClick={handleDecrement} > -1</button>
        <button type="button" class="btn btn-warning"
          onClick={handlePrivacyToggle}>Privacy Toggle</button>
      </div>
      <div className="d-grid gap-2 d-sm-flex justify-content-sm-center
        control-row">
        <input type="text" placeholder="Enter number"
          className="number-input" ref={inputElement} />
        <button type="button" className="btn btn-info"
         onClick={handleAdd}> Add </button>
        <button type="button" className="btn btn-danger"
          onClick={handleSubtract} > Subtract </button>
      </div> </> );}; export default Controls;

index.js //ye store hai
import {createStore} from "redux"; //import kiye store creation ke liye
const INITIAL_VALUE = { counter: 0, privacy: false} //initial value
const counterReducer = (store = INITIAL_VALUE, action) => {
  if (action.type === 'INCREMENT') {
    return {...store, counter: store.counter + 1};
  } else if (action.type === 'DECREMENT') {
    return {...store, counter: store.counter - 1};
  } else if (action.type === 'ADD') {
    return {...store, counter: store.counter +
Number(action.payload.num)};
  } else if (action.type === 'SUBTRACT') {
    return {...store, counter: store.counter -
Number(action.payload.num)};
  } else if (action.type === 'PRIVACY_TOGGLE') {
    return {...store, privacy: !store.privacy};
  }
  return store;
}//reducer function bnaye
const counterStore = createStore(counterReducer); //store create kiye
export default counterStore;
```

Main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import { Provider } from "react-redux";
import counterStore from "./store/index.js";
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <Provider store={counterStore}> //store wala counterStore, me wrapping
      <App />
    </Provider>
  </React.StrictMode>
);
```

57. Why Redux Toolkit

1. Action types are difficult to maintain
2. Store becoming too big
3. Mistakenly editing store
4. Reducer becoming too big

58. Working with Redux Toolkit

1. npm install @reduxjs/toolkit
2. Remove redux from package.json
3. import {createSlice} from "@reduxjs/toolkit"
4. Slices of the store can be created using the following syntax:

```
const slice = createSlice({
name: '', initialState: {},
reducers: { smallReducerMethods: (state, action) => { }, } })
```

5. ConfigureStore combines multiple reducers and can be used as:

```
configureStore({ reducer: {name: slice.reducer} })
```

6. Export actions = slice.actions;
7. Actions can be dispatched like: actions.reducerMethod(payload);

Pichhla same project(Counter App) ko ab Redux Toolkit se bnayenge

App.jsx

No Change

Container.jsx

No Change

Header.jsx

No Change

PrivacyMessage.jsx

No Change

App.css

No Change

DisplayCounter.jsx

```
import { useSelector } from "react-redux"; //import kiye
const DisplayCounter = () => {
  const { counterVal } = useSelector((store) => store.counter);
  return <p className="lead mb-4">Counter current Value: {counterVal}</p>;
}; //counter store se value nikale
export default DisplayCounter;
```

Controls.jsx

```
import { useRef } from "react";
import { useDispatch } from "react-redux";
import { counterActions } from "../store/counter";
import { privacyActions } from "../store/privacy";
const Controls = () => {
  const dispatch = useDispatch(); //useDispatch use kiye
  const inputElement = useRef();
  const handleIncrement= () => {
    dispatch(counterActions.increment());}; //fun body pahle se alag h
  const handleDecrement= () => {
    dispatch(counterActions.decrement());}; //fun body pahle se alag h
  const handlePrivacyToggle= () => {
    dispatch(privacyActions.toggle());}; //fun body pahle se alag h
  const handleAdd= () => {
    dispatch(counterActions.add(inputElement.current.value));
    inputElement.current.value = "";}; //fun body pahle se alag h
  const handleSubtract= () => {
    dispatch(counterActions.subtract(inputElement.current.value));
    inputElement.current.value = "";}; //fun body pahle se alag h
```

```
return ( <> //ye sara same hi h
      <div className="d-grid gap-2 d-sm-flex justify-content-sm-center">
        <button type="button" className="btn btn-primary"
          onClick={handleIncrement} >+1 </button>
        <button type="button" className="btn btn-success"
          onClick={handleDecrement} > -1</button>
        <button type="button" class="btn btn-warning"
          onClick={handlePrivacyToggle}>Privacy Toggle</button>
      </div>
      <div className="d-grid gap-2 d-sm-flex justify-content-sm-center
        control-row">
        <input type="text" placeholder="Enter number"
          className="number-input" ref={inputElement} />
        <button type="button" className="btn btn-info"
         onClick={handleAdd}> Add </button>
        <button type="button" className="btn btn-danger"
          onClick={handleSubtract} > Subtract </button>
      </div> </> );}; export default Controls;

index.js //ye store hai
import {configureStore, createSlice} from "@reduxjs/toolkit"
import counterSlice from "./counter";
import privacySlice from "./privacy";
const counterStore = configureStore({reducer: {
  counter: counterSlice.reducer, //store ka first slice
  privacy: privacySlice.reducer  //store ka 2nd slice
}});export default counterStore;
privacy.js
import { createSlice } from "@reduxjs/toolkit";
const privacySlice = createSlice({
  name: 'privacy',
  initialState: false,
  reducers: {
    toggle: (state) => {
      return state = !state;
    }
  }
});
export const privacyActions = privacySlice.actions;
export default privacySlice;
```

counter.js

```
import { createSlice } from "@reduxjs/toolkit";
const counterSlice = createSlice({
  name: 'counter',
  initialState: {counterVal: 0},
  reducers: {
    increment: (state) => {
      state.counterVal++; },
    decrement: (state) => {
      state.counterVal--; },
    add: (state, action) => {
      state.counterVal += Number(action.payload);},
    subtract: (state, action) => {
      state.counterVal -= Number(action.payload);}}});
export const counterActions = counterSlice.actions;
export default counterSlice;
```

main.jsx

No Change

To Start FrontEnd:

Go to exact folder path:

1. npm install
2. npm run dev

To Start BackEnd:

Go to exact folder path:

1. npm install
2. npm start

Frequently used npm commands:

1. [node --version] & [npm --version]
2. npm create vite@latest [vite@4.4.1], cd React_Radiance
   npm install, npm run dev **[to launch dev server]**
3. **BootStrap**: npm i bootstrap@5.3.3
4. **React-Icons**: npm install react-icons --save
5. **React-Router**: npm install react-router-dom
6. **Redux**: npm init -y, npm install redux
7. **React-with-Redux**: npm install redux, npm install react-redux
8. **Redux-Toolkit**: npm install @reduxjs/toolkit

Project: Myntra Functional Clone (Revise)