React HandWritten Notes

1. What is reactJs :-

i) **React** is **JavaScript Library**
ii) React use to make **web front application**
iii) with **React** we make **SPA(single page application)**

2. Why ReactJs :-

- Fast because of **virtual DOM**
- **Easy to learn**
- **Learning curve** is also **less complex** than other **UI technology**
- **High demand** and **lots of job** in market
- **Large community**

3. More About ReactJs :-

- **Develop** by **Facebook**
- **Current version is 19**
- After **React** you Can easily learn **React Native** also for **mobile application development**

4. Install ReactJs :-

i) Install **create-vite** (if you don't have it):

```
npm create vite@latest
```

ii) Enter your project name and select **React** (or **React + JavaScript** if you prefer JavaScript).

iii) Install **dependencies**:

```
cd your-project-name
npm install
```

iv) Run the **development server**:

```
npm run dev
```

5. Why vite for React Setup :-

- **Vite** is a **fast Development server** and **build tools**
- Efficient **production build** and **simple Configuration**
- **Type Script support**

6. File and Folder Structure :-

```
your-project-name/
├── node_modules/   → Project dependencies
├── public/          →  Static files (images, favicon etc.)
│   └── vite.svg
├── src/             → Source code folder
│   ├── assets/      → Images, icons, fonts
│   ├── App.jsx      → Main React component
│   ├── main.jsx     → Entry point of React app
│   └── index.css    → Global CSS styles
├── .gitignore       →  Git ignore file
├── index.html       → HTML template
├── package.json     → Project metadata & dependencies
├── vite.config.js   → Vite configuration file
└── README.md        → Project documentation
```

7. Your first React Component :-

What is a Component in React ?

- A component is a reusable, self-contained piece of UI (User Interface). It defines how a part of the UI looks and behaves. Think of components like building blocks of your React app.

How to Use Components?

- You create components and then use (render) them inside other components or the main app. Components can be composed to build complex UIs.

Role of Components in a React Application :-

- Break the UI into small, manageable pieces
- Make code reusable and organized
- Help in maintaining and scaling applications
- Allow separation of logic and presentation

How to write your first component in React :-

// this is functional component

Example :-

React HandWritten Notes

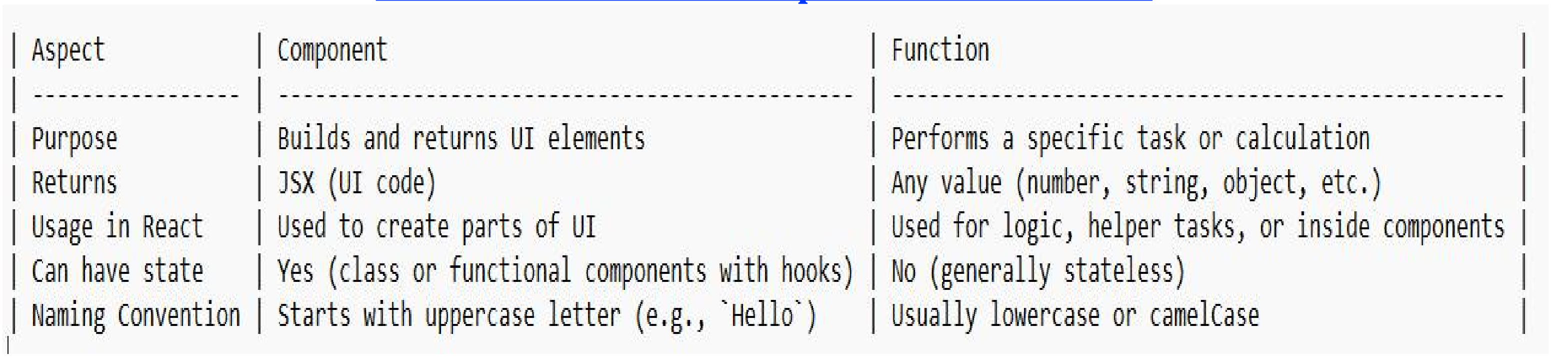
Hello.jsx :-

```
import React from 'react';

function Hello() {
  return <h1>Hello, React!</h1>;
}

export default Hello;
```

App.jsx :-

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <div>
      <Hello />
    </div>
  );
}

export default App;
```

Difference Between Component and Function



Aspect	Component	Function
Purpose	Builds and returns UI elements	Performs a specific task or calculation
Returns	JSX (UI code)	Any value (number, string, object, etc.)
Usage in React	Used to create parts of UI	Used for logic, helper tasks, or inside components
Can have state	Yes (class or functional components with hooks)	No (generally stateless)
Naming Convention	Starts with uppercase letter (e.g., `Hello`)	Usually lowercase or camelCase

Note: ***Virtual DOM*** *is a lightweight copy of the real DOM.*
React updates the Virtual DOM first, compares it with the old one.
Then it updates only the changed parts in the real DOM, making apps faster.

8. importing and exporting components in React :-

Why do we need to import and export components in React?
- To make components **reusable and modular**.

-> **Export** allows a component to be shared with other files.
-> **Import** brings that component into another file where it can be used. This keeps code **organized, clean, and scalable**.

How do you create a new file for a component in React?
- Inside the src/ folder, create a new file (e.g., Hello.jsx)

Write your component inside it
Example:

```
function Hello() {
  return <h1>Hello, React!</h1>;
}
```

How to export a component in React?
Default Export:
Allows one export per file

export default Hello;

Named Export:
Allows multiple exports per file

export { Hello };

Multiple Named Exports in a single file:

```
export function Hello() {
  return <h1>Hello</h1>;
}

export function Greet() {
  return <h2>Welcome!</h2>;
}
```

React HandWritten Notes

How to import a component in React?

Default Import:

import Hello from './Hello';

Named Import:

import { Hello } from './Hello';

Import Multiple Named Components:

import { Hello, Greet } from './Hello';

9. Importance of jsx :-

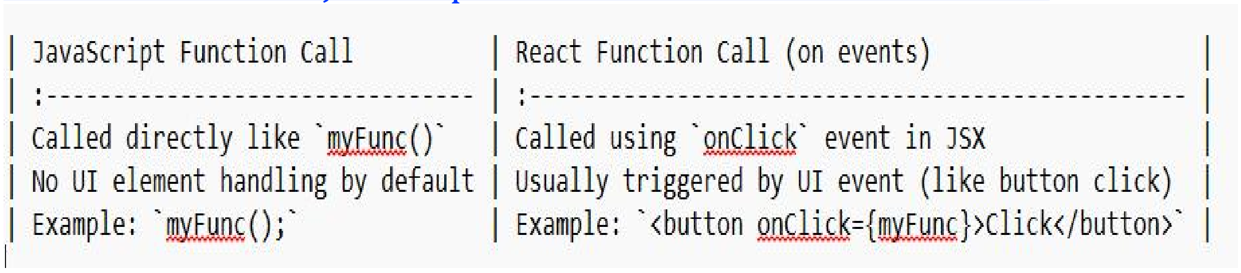
JSX (JavaScript XML) makes it easy to **write HTML inside JavaScript**.

- Helps create UI in a **clear and readable way**
- Lets you combine **HTML and JavaScript logic**
- Makes code **easy to understand and manage**
- React uses JSX to convert code into **virtual DOM** for fast updates

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

10. Click Event & Function call :-

Difference between JavaScript Function Call & React Function Call :-



JavaScript Function Call	React Function Call (on events)
Called directly like `myFunc()`	Called using `onClick` event in JSX
No UI element handling by default	Usually triggered by UI event (like button click)
Example: `myFunc();`	Example: `<button onClick={myFunc}>Click</button>`

How to Make a Function in React :-
function sayHello() {
alert("Hello from React!");
}

How to Make a Button and Click Event :-

```
<button onClick={sayHello}>Click Me</button>
```

How to Call Function on Click Event :-

Simply pass the function name (without ())

```
<button onClick={sayHello}>Click Me</button>
```

If you write onClick={sayHello()} it will call immediately, not on click

How to Call an Arrow Function :-

```
const showMessage = () => {
  alert("Hello from Arrow Function!");
};

<button onClick={showMessage}>Click Here</button>
```

How to Pass Parameters with Function Call :-

```
function greet(name) {
  alert("Hello " + name);
}

<button onClick={() => greet("John")}>Say Hello</button>
```

11. Upgrade React Version :-

```
npm install react@rc

npm install react-dom@rc
```

Use of react@rc :- If you want to experiment with new React features.

12. State & Hooks in ReactJs :-

Why is state required in React?
- State allows React components to **remember and manage data that changes over time**, like user inputs or dynamic content, enabling interactive UIs.

What is state?
- State is a **JavaScript object** that holds data or information about the component. When state changes, the component **re-renders** to reflect the new data.

What are Hooks?
- Hooks are special **functions introduced in React 16.8** that let you use state and other React features inside **functional components** without writing classes.

How to use state in React ?
- Use the **useState** hook inside a functional component to add state.

Example :-

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

Using Multiple States in One Logic Example:

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

function UserForm() {
  const [name, setName] = useState('');
  const [age, setAge] = useState('');

  function handleSubmit() {
    alert(`Name: ${name}, Age: ${age}`);
  }

  return (
    <div>

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

      <input
        type="number"
        placeholder="Enter age"
        value={age}
        onChange={e => setAge(e.target.value)}
      />

      <button onClick={handleSubmit}>Submit</button>

    </div>
  );
}
```

13. Conditional Rendering In React :-

Conditional rendering means showing or hiding parts of the UI based on certain conditions (like a state value or a variable).
It works just like **if-else statements in JavaScript**, but inside JSX.

1.Using if statement :-

```
if (isLoggedIn) {
  return <p>Welcome User!</p>;
} else {
  return <p>Please Login</p>;
}
```

2. Using Ternary Operator (most common)

```
<p>{isLoggedIn ? 'Welcome User!' : 'Please Login'}</p>
```

3. Using && (And) Operator
If the condition is true, show the content

```
{isLoggedIn && <p>Welcome User!</p>}
```

4. Using switch statement (if multiple conditions)

```
switch (status) {
  case 'loading':
    return <p>Loading...</p>;
  case 'error':
    return <p>Error!</p>;
  default:
    return <p>Done!</p>;
}
```

14. Toggle (Hide & Show) in React :-

1.Define State
Use the **useState** hook to create a state for visibility.

```
const [isVisible, setIsVisible] = useState(true);
```

2. Update State on Button Click
Make a function to **change the state value**.

```
function toggleContent() {
  setIsVisible(!isVisible); // toggles true or false
}
```

3.Add Condition for Toggle
Use conditional rendering to decide when to show content.

```
{isVisible && <p>This is a toggleable content!</p>}
```

4.Hide and Show Component Complete example combining all steps:

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

function ToggleExample() {
  const [isVisible, setIsVisible] = useState(true);

  function toggleContent() {
    setIsVisible(!isVisible);
  }

  return (
    <div>
      <button onClick={toggleContent}>
        {isVisible ? 'Hide' : 'Show'} Content
      </button>

      {isVisible && <p>This is a toggleable content!</p>}
    </div>
  );
}

export default ToggleExample;
```

15. Multiple conditional in React :-

1.Define State and Button

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

function MultipleCondition() {
  const [status, setStatus] = useState('idle');
```

And create buttons to change state:

```
return (
    <div>
      <button onClick={() => setStatus('loading')}>Loading</button>
      <button onClick={() => setStatus('success')}>Success</button>
      <button onClick={() => setStatus('error')}>Error</button>
    </div>
   );
}
```

2. Change State Value on Button Click
When a button is clicked, it sets the state to a new value ('**loading**', '**success**', '**error**').

3. Apply Condition with State
Add multiple conditions based on status value:

```
<div>
 {status === 'idle' && <p>Status: Idle</p>}
 {status === 'loading' && <p>Loading, please wait...</p>}
 {status === 'success' && <p>Success! ✅</p>}
 {status === 'error' && <p>Error occurred ❌</p>}
</div>
```

Note : Make Complete Example by your self following above content

16. Props in ReactJs :-

1. What is Props?
Props (Properties) are used to **pass data from one component to another.**
They are **read-only** and passed from **parent to child component** like function arguments.

2. Make a Component
Child Component:

```
function Greeting(props) {
 return <h2>Hello, {props.name}</h2>;
}
```

3.Pass Data Between Components

i) Pass a Variable :-

```
<Greeting name="John" />
```

ii) Pass an Object :-

```
<Greeting user={{ name: 'John', age: 25 }} />
```

iii) Pass an Array :-

```
<Greeting items={['Apple', 'Banana', 'Orange']} />
```

4.Receive and Display Data

Child Component for Object & Array

```
function Greeting(props) {
  return (
    <div>
      <h2>Hello, {props.user.name}</h2>
      <p>Age: {props.user.age}</p>
      <ul>
        {props.items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
}
```

5.Pass Data in Components on Click

Parent Component :-

```
function App() {
  function showMessage(message) {
    alert(message);
  }

  return <Button clickAction={() => showMessage('Hello from Props!')}
/>;
}
```

Child Component :-

```
function Button(props) {
  return <button onClick={props.clickAction}>Click Me</button>;
}
```

17. Get input field value :-

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

function GetInput() {
 // Define state to store input value
 const [name, setName] = useState('');

 // Function to clear the input field
 function clearInput() {
  setName('');
 }

 return (
  <div>
   {/* Make input field & get input value in state */}
   <input
    type="text"
    placeholder="Enter your name"
    value={name}
    onChange={(e) => setName(e.target.value)}
   />

   {/* Display the value from state */}
   <p>Your name is: {name}</p>

   {/* Button to clear input value */}
   <button onClick={clearInput}>Clear</button>
  </div>
 );
}

export default GetInput;
```

OUTPUT :-

- Type anything in the **input box** — it will show below live.
- Click **Clear** button — it will remove the text from input and below.

18. Controlled Component :-

1. What is a Controlled Component?

- A **Controlled Component** is an input element whose value is **controlled by React state**.
-The value inside the input box comes from the state, not directly from the DOM.

2. How to Identify it's a Controlled Component?

- If an input has a **value** set from **state** and uses **onChange** to update that state — it's a **Controlled Component**.

Example:

<input value={name} onChange={(e) => setName(e.target.value)} />

3.Error if You Don't Use value in Controlled Input :-

i) React may warn:
"**A component is changing an uncontrolled input to be controlled**"
ii)This happens if you don't set the value and later try to control it through state.

Fix: Always provide value property from state when controlling.

4. Make a Form & Get Input Field Values

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

function ControlledForm() {
  // Define states for form inputs
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  return (
    <div>
      {/* Controlled Input: value from state + onChange */}
      <input
        type="text"
        placeholder="Enter name"
    value={name}
    onChange={(e) => setName(e.target.value)}
   />

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

   {/* Display entered values */}
   <p>Name: {name}</p>
   <p>Email: {email}</p>
  </div>
 );
}

export default ControlledForm;
```

19. Handle CheckBox :-

1. Make Checkbox :-

<input type="checkbox" />

2.Define State for Checkbox :-

Use useState to store the checkbox state (checked or not):

```
const [agree, setAgree] = useState(false);
```

3. Get Checkbox Value in State :-

Use onChange to update state based on checkbox status:

```
<input
 type="checkbox"
 checked={agree}
 onChange={(e) => setAgree(e.target.checked)}
/>
```

4. Remove Checkbox Value from State :-

You can set the state back to false to uncheck:

```
function uncheck() {
  setAgree(false);
}
```

Full Example

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

function CheckboxHandler() {
  // Define state for checkbox
  const [agree, setAgree] = useState(false);

  // Function to uncheck
  function uncheck() {
    setAgree(false);
  }

  return (
    <div>
      {/* Checkbox controlled by state */}
      <input
        type="checkbox"
        checked={agree}
        onChange={(e) => setAgree(e.target.checked)}
      />
      <label>I Agree</label>

      {/* Show checkbox value */}
      <p>Checked: {agree ? 'Yes' : 'No'}</p>

      {/* Button to uncheck */}
      <button onClick={uncheck}>Uncheck</button>
    </div>
  );
}

export default CheckboxHandler;
```

OUTPUT :-

- Tick checkbox → shows Checked: Yes
- Untick → shows Checked: No
- Click **Uncheck** button → checkbox gets unticked

20. Handle Radio and Dropdown :-

```
// 1. Import useState hook from react
import { useState } from "react";

// 2. Define functional component App
function App() {

 // 3. Declare state variables
 const [gender, setGender] = useState("");
 const [city, setCity] = useState("");

 // 4. Return JSX UI
 return (
  <div>
   {/* 5. Heading */}
   <h1>Handle Radio button & DropDown</h1>

   {/* 6. Gender Radio Buttons */}
   <h3>Select Gender :</h3>

   {/* Male Radio Button */}
   <input
    type="radio"
    onChange={(e) => setGender(e.target.value)} // Update state on
change
    name="gender"
    id="male"
    value={"Male"}
    checked={gender == "Male"} // Controlled checked
   />
   <label htmlFor="male">Male</label>
```

```
    {/* Female Radio Button */}
    <input
     type="radio"
     onChange={(e) => setGender(e.target.value)} // Update state on change
     name="gender"
     id="female"
     value={"Female"}
     checked={gender == "Female"} // Controlled checked
    />
    <label htmlFor="female">Female</label>

    {/* 7. Display Selected Gender */}
    <h2>Selected Gender: {gender}</h2>

    <br /> <br />

    {/* 8. City Dropdown */}
    <h3>Select City :</h3>
    <select
     defaultValue={"delhi"} // Set default selected option
     onChange={(e) => setCity(e.target.value)} // Update state on
selection
    >
     <option value="noida">Noida</option>
     <option value="delhi">Delhi</option>
     <option value="bharuch">Bharuch</option>
    </select>

    {/* 9. Display Selected City */}
    <h3>Selected City: {city}</h3>
   </div>
  );
}

// 10. Export App component as default
export default App;
```

OUTPUT:

- Select **Male/Female** → shows selected gender.
- Choose **city** from dropdown → shows selected city.

21. Loop in JSX with Map Function :-

1. What is an Array?

- A collection of multiple values stored in a single variable.
Example:

```
const fruits = ["Apple", "Banana", "Mango"];
```

2. Make an Array :-

- Let's create an array of objects:

```
const students = [
 { id: 1, name: "Raj" },
 { id: 2, name: "Simran" },
 { id: 3, name: "Aman" }
];
```

3. Make a Table in JSX :-

-Basic table structure:

```
<table>
 <thead>
  <tr>
   <th>ID</th>
   <th>Name</th>
  </tr>
 </thead>

 <tbody>
  {/* Data rows will come here */}
 </tbody>

</table>
```

4.Use map() Function for Looping :-

- map() loops over array items and returns JSX for each.
 Example:

```
<tbody>
```

```
 {students.map((item) => (
   <tr key={item.id}>
     <td>{item.id}</td>
     <td>{item.name}</td>
   </tr>
 ))}
</tbody>
```

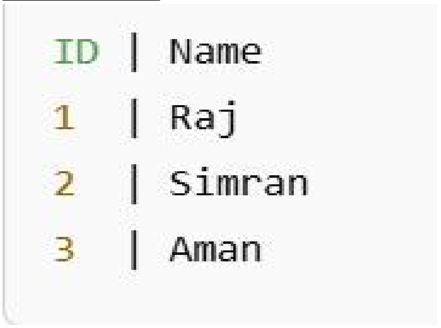
=> key={item.id} is a unique key for React list items.

Full Example Code:

```
import React from "react";

function App() {
 // Array of students
 const students = [
   { id: 1, name: "Raj" },
   { id: 2, name: "Simran" },
   { id: 3, name: "Aman" }
 ];

 // Table and  map() loop
 return (
   <div>
     <h1>Students List</h1>
     <table border={1}>
       <thead>
         <tr>
           <th>ID</th>
           <th>Name</th>
         </tr>
       </thead>
       <tbody>
         {/* Loop through students array */}
         {students.map((item) => (
           <tr key={item.id}>
             <td>{item.id}</td>
             <td>{item.name}</td>
           </tr>
         ))}
```

```
    </tbody>
   </table>
  </div>
 );
}

export default App;
```

OUTPUT:



```
ID | Name
1  | Raj
2  | Simran
3  | Aman
```

22. Reuse Component :-

Make the **Student.jsx** Component (as a Table Row)

Student.jsx :-

```
function Student(props) {
 return (
  <tr>
   <td>{props.id}</td>
   <td>{props.name}</td>
   <td>{props.city}</td>
  </tr>
 );
}

export default Student;
```

Use **map()** in **App.jsx** to Render Table Rows

App.jsx :-

```
import Student from "./Student";

function App() {
 //1. Array of student objects
 const students = [
  { id: 1, name: "Raj", city: "Delhi" },
  { id: 2, name: "Simran", city: "Noida" },
  { id: 3, name: "Aman", city: "Bharuch" },
 ];

 return (
  <div>
   <h1>Student List  (in Table)</h1>

   {/*2. Table Header */}
   <table border="1" cellPadding="10" cellSpacing="0">
    <thead>
     <tr>
      <th>ID</th>
      <th>Name</th>
      <th>City</th>
     </tr>
    </thead>
     {/* Table Body */}
    <tbody>
     {/* 3. Render Student component in loop */}
     {students.map((student) => (
      <Student
       key={student.id}
       id={student.id}
       name={student.name}
       city={student.city}
      />
     ))}
    </tbody>
   </table>
  </div>
 );
}
export default App;
```

23. Nested Looping :-

```
import React from 'react';

// Sample 2D array (array of arrays)
const data = [
  ['A1', 'A2', 'A3'],
  ['B1', 'B2', 'B3'],
  ['C1', 'C2', 'C3']
];

// Component for inner loop item
function InnerItem({ item }) {
  // Render a single item with some margin for spacing
  return <span style={{ margin: '5px' }}>{item}</span>;
}

// Component for outer loop row
function OuterRow({ row }) {
  return (
    <div>
      {
        // Inner loop: map through each element in the row
        row.map((item, idx) => (
          // Render InnerItem component for each element
          <InnerItem key={idx} item={item} />
        ))
      }
    </div>
  );
}

// Main component that contains nested loops
export default function NestedLoopExample() {
  return (
    <div>
      {
        // Outer loop: map through each row in the data array
        data.map((row, idx) => (
          // Render OuterRow component for each row
          <OuterRow key={idx} row={row} />
        ))
      }
    </div>
  ); }
```

24. Hooks in React :-

1. What are Hooks ?

- **Hooks are functions in React that let us use state and other features in functional components.**
- Before Hooks, only class components could have state and lifecycle methods.

2. Why We Need Hooks?

because of:

- Earlier, only class components could handle state.
- Hooks make function components powerful and cleaner.
- They help us reuse logic easily using custom hooks.

3. History of Hooks :-

- **Introduced in React v16.8 (February 2019)**
- Announced at **React Conf 2018**
- Designed to bring state and other React features (like context, lifecycle features) to functional components.
- React team introduced them to **simplify complex class components** and improve code reusability.

4. Some Common Hooks :-

Hook Name	Use
useState	*For adding state in function components*
useEffect	*For side effects (API calls, subscriptions)*
useContext	*To use React Context API*
useRef	*To reference DOM elements or store mutable values*
useMemo	*For memoizing expensive calculations*
useCallback	*Memoizes callback functions*
useReducer	*For complex state management (like Redux)*
useLayoutEffect	*Similar to useEffect but fires synchronously*
useImperativeHandle	*Customizing instance values exposed by ref*

5. How to Identify Hooks?
Hooks are JavaScript functions whose names start with **use**
Example: useState, useEffect, useContext
Custom hooks also **must start** with **use** -> useFetchData(), useAuth(), etc.

Rules for using hooks:

Call Hooks **at the top level** of your function (not inside loops, conditions, or nested functions)

Only call Hooks from:

- **React function components**
- **Custom hooks**

25. useEffect Hook :-

1.What is the use of useEffect?
- useEffect is a React Hook used to **handle side effects** in a function component.
- Side effects: things **like API calls, setting timers, updating DOM, or changing state after render**.
- It runs **after every render by default.**

2.What Example Will We Take for This?
Simple Example:

- Changing the document title when a button is clicked.
- Fetching data from an API when a component mounts.

3. Syntax of useEffect :-

```
useEffect(() => {
  // your code here (side effect)
}, [dependencies]);
```

-> The **first argument** is a callback function.
-> The **second argument** is an array of dependencies.

4. Handling Dependancy :-

```
useEffect(() => {
  console.log("Runs every time");
}); // no dependency

useEffect(() => {
  console.log("Runs only once");
}, []); // empty dependency

useEffect(() => {
  console.log("Runs when 'count' changes");
}, [count]); // single state

useEffect(() => {
  console.log("Runs when 'count' or 'value' changes");
}, [count, value]); // multiple states

useEffect(() => {
  console.log("Runs when 'name' prop changes");
}, [name]); // prop dependency
```

5. useEffect with State:-

Example:

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

function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("Count changed:", count);
  }, [count]);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Click
{count}</button>
    </div>
  );
}
```

6. useEffect with Props :-

Example:

```
function WelcomeMessage({ name = "xyz" }) {
  useEffect(() => {
    console.log("Name changed:", name);
  }, [name]);

  return <h1>Welcome {name}</h1>;
}
```

Here, useEffect runs when the name prop changes.

26. Component life Cycle :-

1.What is Human Life Cycle?

A human life cycle is the series of stages a person goes through in life:
Born → Grows → Lives → Dies

2. What is Component Life Cycle in React?

A component life cycle is the series of stages a React component goes through:
Mounting → Updating → Unmounting

3. Phase of life cycle :-

Phase	Description
Mounting	When the component is being created and inserted into the DOM
Updating	When the component is updated (state or props changes)
Unmounting	When the component is removed from the DOM

4. How to Use useEffect to Handle Life Cycle?

Life Cycle Phase	How to Handle it with useEffect
Mounting	useEffect(() => { /* code */ }, [])
Updating	useEffect(() => { /* code */ }, [state/props])
Unmounting	useEffect(() => { return () => { /* cleanup code */ } }, [])

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

function Demo() {
  const [count, setCount] = useState(0);

  // Mounting
  useEffect(() => {
    console.log("Component Mounted");
  }, []);

  // Updating (on count change)
  useEffect(() => {
    console.log("Count updated:", count);
  }, [count]);

  // Unmounting
  useEffect(() => {
    return () => {
      console.log("Component will unmount");
    };
  }, []);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Click
{count}</button>
    </div>
  );
}
```

27. useRef Hook In React

1.What is useRef Hook in React?

useRef is a React Hook that lets you persist a mutable value across renders **without causing a re-render when you change it**.

Typical use cases:

- Storing a reference to a DOM element.
- Storing mutable variables that don't trigger re-renders.
- Keeping track of previous values.

2.Control an Input Field using useRef Hook :-

You can use **useRef** to directly access an input DOM node and control its value without using state.

Example :

```
function InputControl() {
  const inputRef = useRef(null);

  const handleClick = () => {
    // set input value
    inputRef.current.value = "Hello, this is useRef!";
    inputRef.current.style.color ="Blue" ;
    // focus the input
    inputRef.current.focus();
  };

  return (
    <div>
      <input type="text" ref={inputRef} placeholder="Enter Your Name:" />
      <button onClick={handleClick}>Set Text & Focus</button>
    </div>
  );
}

export default InputControl;
```

28. Uncontrolled Compoent

1.What is an Uncontrolled Component in React?

In React, a form component is **uncontrolled** when the form data is handled by the **DOM itself** rather than React state.
You access the form value using **refs** or **document query selectors** instead of React's useState.

In short:
-> Data is handled by the DOM.
-> You read the data when needed (for example, when submitting a form).

2.Make an Uncontrolled Component with querySelector

You can directly use **document.querySelector** to grab the DOM element and get its value — which is technically possible, though it's not the "React way". It works fine in small cases.

Example:

```
import React from 'react';

function UncontrolledWithQuerySelector() {
  const handleSubmit = (e) => {
    e.preventDefault();
    const inputValue = document.querySelector('#myInput').value;
    alert('Input Value: ' + inputValue);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" id="myInput" />
      <button type="submit">Submit</button>
    </form>
  );
}

export default UncontrolledWithQuerySelector;
```

3.Make an Uncontrolled Component with useRef :-

This is the clean React way to make an uncontrolled component — using **useRef** to access the DOM input directly.

Example:

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

function UncontrolledWithUseRef() {
  const inputRef = useRef(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    alert('Input Value: ' + inputRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" ref={inputRef} />
      <button type="submit">Submit</button>
    </form>
  );
}

export default UncontrolledWithUseRef;
```

29. Pass a Function as Props in React

App.js :- (Parent Component)

```
import React from 'react';
import Child from './Child';

function App() {
  // Function to pass as props
  const showMessage = () => {
    alert('Hello from the Parent Component!');
  };
```

```
 return (
   <div>
     <h1>React Function as Props Example</h1>
     <Child handleClick={showMessage} />
   </div>
 );
}

export default App;
```

Child.js :- (Child Component)

```
import React from 'react';

function Child(props) {
 return (
   <div>
     <button onClick={props.handleClick}>Click Me</button>
   </div>
 );
}
export default Child;
```

30. forwardRef Hook

1.What is forwardRef in React ?

- **forwardRef** is a React function that lets you pass a ref from a parent component to a child component's DOM element.

Normally, you can only attach ref to a DOM element, not to a function component directly.

2.Why do we use forwardRef ?

- To let the **parent component control or access the DOM element inside the child component**.

For example:

- Focus an input
- Scroll a div
- Get value or size of an element
- Integrate with libraries needing DOM nodes

CustomInput.jsx :-

```
import React, { forwardRef } from "react";

function CustomInput(props, ref) {
  return <input ref={ref} {...props} />;
}

export default forwardRef(CustomInput);
```

App.jsx :-

```
import React, { useRef } from "react";
import CustomInput from "./CustomInput";

function App() {
  const inputRef = useRef();

  const handleFocus = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <CustomInput ref={inputRef} placeholder="Type here..." />
      <button onClick={handleFocus}>Focus the Input</button>
    </div>
  );
}

export default App;
```

31. useFormStatus

1.What is useFormStatus in React?

- useFormStatus is a React hook introduced in React 18.2+ with the experimental React Server Components (RSC) API — specifically for handling form submission status inside a form action.

It lets you easily check:
- If the form is submitting
- Handle pending state
- Show loading indicators
- Disable inputs/buttons during submission

It works with React's <form action={someFunction}> method (React Server Actions).

Example :

```
import React from "react";
import { useFormStatus } from "react-dom";

function SubmitButton() {
  const { pending } = useFormStatus();

  return (
    <button type="submit" disabled={pending}>
      {pending ? "Submitting..." : "Submit"}
    </button>
  );
}

function App() {
  async function handleFormSubmit(formData) {
    // simulate server-side work
    await new Promise((resolve) => setTimeout(resolve, 2000));
    console.log("Form Data:", {
      name: formData.get("name"),
      email: formData.get("email"),
    });
  }

  return (
    <form action={handleFormSubmit}>
      <div>
        <label>Name:</label>
        <input name="name" type="text" required />
      </div>
      <div>
        <label>Email:</label>
      <input name="email" type="email" required />
    </div>
    <SubmitButton />
  </form>
 );
}

export default App;
```

32. useTransition Hooks

useTransition is a React hook used to mark some state updates as non-urgent (low-priority), so the UI stays responsive for urgent updates like clicks, typing, etc.

It's helpful when you have heavy or slow state updates (like filtering a big list) — and you don't want them to block immediate interactions.

Example :

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

const App = () => {
 const [pending, setTransition] = useTransition(); // get pending
state and setTransition function

 const handleSubmit = () => {
  setTransition(async () => {
    await new Promise(res => setTimeout(res, 2000)); // simulate a 2
sec delay
  });
 }

 return (
  <div>
    <h1>This is useTransition Example</h1>
    <button disabled={pending}
onClick={handleSubmit}>Submit</button>
```

```
    {
    pending?
    <img
src="https://upload.wikimedia.org/wikipedia/commons/b/b1/Load
ing_icon.gif" alt="" />
    :null
    }
   </div>
  )
}

export default App;
```

33. Lifting State up

What is Lifting State Up in React?
- Lifting State Up means moving shared state from child components to their closest common parent component so both (or multiple) components can access and control it.

In React — state should live in the component that needs to control it, or the nearest common ancestor if multiple components need to share it.

Why Do We Lift State Up?
- To keep data consistent between components
- To allow sibling components to communicate
- To manage shared logic or values centrally

Scenario:
Two components:

InputBox → lets user type text
DisplayText → shows the text

Both need to access the same text state

Exmaple :

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

function App() {
  const [text, setText] = useState("");

  return (
    <div>
      <InputBox text={text} setText={setText} />
      <DisplayText text={text} />
    </div>
  );
}

function InputBox({ text, setText }) {
  return <input value={text} onChange={(e) => setText(e.target.value)}
/>;
}

function DisplayText({ text }) {
  return <p>{text}</p>;
}

export default App;
```

34. useAction Hook

- use to handle **from** in **Reactjs**
- It **updates state** based on the **results of a from action**

Example :

```
import { useActionState } from "react";

const Action = () => {
    const handleSubmit = async(previousData, formData) =>{
        let name = formData.get('name');
        let password = formData.get('password');

        await new Promise(res=>setTimeout(res,2000));
        if(name && password){
           return {data:'Data submitted',name,password};
        }
        else{
           return{error:'Failed form',name,password};
        }

    }
    const[data,action,pending] =
useActionState(handleSubmit,undefined);
  return (
    <div>
        <form action={action}>
           <input defaultValue={data?.name} type="text"
placeholder='Enter Name:' name='name' /><br /><br />
           <input defaultValue={data?.password} type="text"
placeholder='Enter Password' name='password' /><br /><br />
           <button disabled={pending}>Submit Data</button><br />
           {
              data?.error && <span>{data?.error}</span>
           }
           {
              data?.data && <span>{data?.data}</span>
           }
        </form>
        <h2>Name:{data?.name}</h2>
        <h2>Password:{data?.password}</h2>
    </div>
  )
}

export default Action;
```

35. useID Hook

1.What is useId Hook in React?

- useId is a built-in React Hook introduced in React 18.

Purpose:

It generates a unique, stable ID string that can be used for accessibility attributes like id, htmlFor, or linking elements inside the

DOM (especially useful when rendering multiple components and needing unique IDs that stay the same between server and client rendering — useful in SSR/React hydration).

-> It's a stable unique ID for the life of the component.

2.How to Use useId

Syntax:

```
const id = useId();
```

Example :

```
import React, { useId } from "react";

function NameForm() {
  const nameInputId = useId();

  return (
    <div>
      <label htmlFor={nameInputId}>Name:</label>
      <input type="text" id={nameInputId} />
    </div>
  );
}

export default NameForm;
```

36. Fragement in ReactJs

1.What is a Fragment in React?

- A Fragment is a special type of component in React that lets you group multiple elements without adding extra nodes to the DOM.

-> In normal cases, if you want to return multiple elements from a component, you have to wrap them inside a parent element like <div> — but sometimes you don't want that extra <div> in your final HTML.

->That's where Fragments come in.

2. Why use Fragments? (Without issues) :-

Benefits / Avoid issues like:

- Avoid extra/empty <div> wrappers (called div soup).
- Keep the DOM clean and optimized.
- Maintain valid HTML structure.
- Helpful in cases like table rows (<tr>) where adding a parent <div> is invalid inside a table.

3. How to Use Fragments :-

I) React.Fragement ii) <>... </>

Example :

```
function Demo() {
  return (
    <div>
      <h1>Hello</h1>
    </div>
    <div>
      <p>This will cause an error</p>
    </div>
  );
}
```

37. Context API in React

1. What is context API ?

The **Context API** in React is a built-in way to **pass data through the component tree without having to pass props down manually at every level.**

It is useful for managing **global state** like user authentication status, themes, language settings, etc., that multiple components need to access.

2.How Does Context API Work?

It involves three main steps:

Create Context: Using **React.createContext()**

Provide Context:Using a **<Context.Provider>** to wrap the component tree and pass the data.

Consume Context: Using **useContext()** hook or Context.Consumer to access the data in child components.

Example :

```
import React, { createContext, useContext } from 'react';

// Step 1: Create Context
const UserContext = createContext();

// Step 2: Provide Context
const App = () => {
  const user = { name: "John Doe", age: 25 };

  return (
    <UserContext.Provider value={user}>
      <UserProfile />
    </UserContext.Provider>
  );
};

const UserProfile = () => {
  // Step 3: Consume Context
  const user = useContext(UserContext);

  return (
    <div>
      <h1>Name: {user.name}</h1>
      <h2>Age: {user.age}</h2>
    </div>
  );
};

export default App;
```

38. Custom Hook

1.What is a Custom Hook in React?

A custom hook is a JavaScript function whose name starts with use and that can call other hooks inside it.

Why Custom Hooks?

- To extract and reuse logic across multiple components.
- To keep components clean and separate logic concerns.
- To follow the DRY principle (Don't Repeat Yourself).

They work just like built-in hooks (like useState, useEffect) but are made by you for your needs.

useToggle.jsx :-

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

const useToggle = (initialValue = false) => {
  const [value, setValue] = useState(initialValue);

  const toggle = () => {
    setValue(prev => !prev);
  };

  return [value, toggle];
};

export default useToggle;
```

App.jsx :

```
import React from 'react';
import useToggle from './useToggle';

const ToggleComponent = () => {
  const [isVisible, toggleVisibility] = useToggle();

  return (
    <div style={{ padding: '20px' }}>
      <button onClick={toggleVisibility}>
        {isVisible ? 'Hide' : 'Show'} Text
    </button>

    {isVisible && <p>This text is toggled!</p>}
  </div>
 );
};

export default ToggleComponent;
```

39. React Router

1.What is React Router?

- React Router is a popular routing library for React applications. It allows your app to have multiple pages (routes) with navigation between them — all without a full page reload. It controls what should be displayed on the screen based on the current URL.

Example:
If your app has:

/home → shows the Home component

/about → shows the About component

React Router helps manage this smoothly using a component-based approach.

2. How to Install React Router (React 19 compatible)

React 19 uses the same router package as React 18 — the latest version is React Router DOM v6.

Install with npm:
npm i react-router

Install react-router :-
npm install react-router-dom

Example :

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

function Home() {
  return <h2> Home Page</h2>;
}

function About() {
  return <h2>About Page</h2>;
}

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

Summary :

- **<BrowserRouter /> :-** This component enables client-side routing using browser's history.
- **<Routes /> :-** It's responsible for rendering the appropriate component based on the current URL.
- **< Route /> :-** Each route component defines a path and the corresponding component to render when that path is matched.
- **< Link /> :-** A Link for navigate from 1 page to other page

40. Page not found and 404 Page and Redirection

1.What is a 404 Page?

A 404 page is a special page shown to users when they visit a URL route in your app that doesn’t exist.
In React Router, you typically handle this with a Route using a wildcard path * to catch any undefined routes.

2.Make a Route for 404 Page :-

In your <Routes> list, the wildcard path="*" will catch all unmatched URLs and render your 404 page component.

3.Example:

```
import { Route, Routes } from "react-router";
import About from "./about";
import Home from "./Home";
import Login from "./Login";
import Nav from "./Nav";
import NotFound from "./NotFound";

function App() {
  return (
    <>
      <Nav />
      <Routes>
        <Route path="/" element={<Home />}/>
        <Route path="/about" element={<About />} />
        <Route path="/login" element={<Login />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </>
  );
}

export default App;
```

4.Redirect From 404 (Optional) :-

If instead of showing a custom page, you want to automatically redirect to Home or another page when a wrong URL is hit — you can use Navigate.

Example: Redirect to / when route is not found :-

```
import { Navigate } from 'react-router-dom'

<Route path="*" element={<Navigate to="/" />} />
```

41. Nested Navigation with React router Example

1.What is Nested Navigation?

- Nested Navigation means having sub-routes (child routes) inside a main route.

For example:

/about/team
/about/contact

Both are pages under the /about section.
In React Router v6+, this is done by nesting <Route> inside a parent route’s element which includes an <Outlet /> — a placeholder where the child pages will render.

Example :

```
import { Link, Outlet } from 'react-router-dom'

function About() {
  return (
    <div className="p-5">
      <h1 className="text-3xl font-bold mb-4">About Page</h1>
      <nav className="flex space-x-4 mb-4">
        <Link to="team" className="text-blue-500 underline">Team</Link>
        <Link to="contact" className="text-blue-500 underline">Contact</Link>
      </nav>
      <Outlet /> {/* Renders nested route content here */}
    </div>
  )
}
export default About
```

App.jsx :

```
<Route path="/about" element={<About />}>
        <Route path="team" element={<Team />} />
        <Route path="contact" element={<Contact />} />
</Route>
```

42. Layout and Index Routes

1.What is a Layout Route?

- A Layout Route is a React Router route that renders common layout content for a group of child routes — like a shared navbar, footer, sidebar, etc.

- It uses an <Outlet /> component as a placeholder where the child routes render their specific content.

Example:

```
function Layout() {
  return (
    <div>
      <Navbar />
      <Outlet />
      <Footer />
    </div>
  )
}
```

2.What is an Index Route?

- An Index Route is like the "default child route" inside a parent route. When you visit a parent path (like /about), the index route's element renders inside the parent's <Outlet />.

It's defined by adding index to the <Route>.

Example:

```
<Route path="/about" element={<About />}>
  <Route index element={<AboutWelcome />} />
  <Route path="team" element={<Team />} />
</Route>
```

Visiting /about shows <AboutWelcome />

Visiting /about/team shows <Team />

3. Why do we use these?

i)Layout Route:

- To avoid repeating navbar/footer/sidebar code in every page.
- To provide a common structure for nested routes.

ii) Index Route:

- To display a default page when visiting a parent path (like /about), before clicking to a nested page.

4.What is a Route Prefix?

- A route prefix is basically when a route path includes a parent path, and its child routes inherit that prefix in their URLs.

For example:

```
<Route path="about" element={<About />}>
  <Route path="team" element={<Team />} />
</Route>
```

Here:

- /about is the parent route

- /about/team is the prefixed child route

43. Dynamic Routes

1.What is a Dynamic Route?

- A dynamic route is a route where part of the URL is a variable or parameter, often represented by a colon : in the path.

Example:

```
<Route path="user/:id" element={<UserDetail />} />
```

Here :id is a placeholder for a dynamic value (like a user's id).

2.Make a User List Page :-

UserList.jsx

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

function UserList() {
  const users = [
    { id: 1, name: "John Doe" },
    { id: 2, name: "Alice Smith" },
    { id: 3, name: "Bob Johnson" }
  ];

  return (
    <div>
      <h2 className="text-2xl">User List</h2>
      <ul className="space-y-2 mt-4">
        {users.map((user) => (
          <li key={user.id}>
            <Link        to={`/user/${user.id}`}        className="text-blue-500
underline">
              {user.name}
            </Link>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default UserList;
```

3. UserDetail.jsx

```
import { useParams } from "react-router-dom";
```

```
function UserDetail() {
  const { id } = useParams();

  return (
    <div>
      <h2 className="text-2xl">User Detail</h2>
      <p className="mt-2">Showing details for user ID: {id}</p>
    </div>
  );
}

export default UserDetail;
```

Set Up Dynamic Routing :-

```
import { Routes, Route } from "react-router-dom";
import Nav from "./Nav";
import Home from "./Home";
import UserList from "./UserList";
import UserDetail from "./UserDetail";
import NotFound from "./NotFound";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Nav />}>
        <Route index element={<Home />} />
        <Route path="users" element={<UserList />} />
        <Route path="user/:id" element={<UserDetail />} />
        <Route path="*" element={<NotFound />} />
      </Route>
    </Routes>
  );
}

export default App;
```

44. Optional Segment

1.What is an Optional Segment?

- An optional segment is a part of a URL path that might or might not be present.
- In React Router, optional segments are handled with multiple routes because it doesn't directly support optional segments with a ? like some routers — but you can easily achieve it using nested or multiple route definitions.

2. Static Optional Segment

- A static optional segment means you might optionally include a static part of the URL.

Example:

```
<Route path="profile" element={<Profile />} />
<Route path="profile/settings" element={<Settings />} />
```

- /profile → shows Profile
- /profile/settings → shows Settings

Both work because settings is an optional extension to the static profile path.

3.Dynamic Optional Segment :-

- A dynamic optional segment would be a parameter like an id or slug that might or might not be present.
- Since React Router v6 doesn't allow syntax like :id? directly in the path, the best practice is to create two separate routes:

Example:

```
<Route path="product" element={<ProductList />} />
<Route path="product/:id" element={<ProductDetail />} />
```

- /product → shows ProductList
- /product/123 → shows ProductDetail with id = 123

Then in your navigation:

```
<Link to="/product">All Products</Link>
<Link to="/product/101">Product 101</Link>
```

And in **ProductDetail.jsx:**

```
import { useParams } from "react-router-dom";

function ProductDetail() {
  const { id } = useParams();

  return (
    <div>
      <h2>Product Detail</h2>
      <p>{id ? `Product ID: ${id}` : "No product selected."}</p>
    </div>
  );
}
```

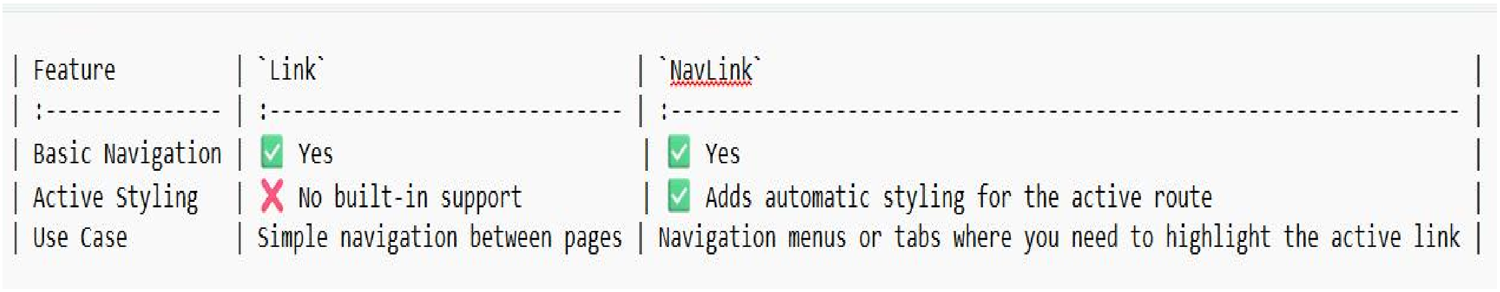
45. NavLink and Active class

1.What is NavLink?

- NavLink is a special version of the Link component in React Router that can apply a specific style or class when the link is active (matches the current URL).

It's mainly used for navigation menus, tabs, or anywhere you want to visually indicate which page you're currently on.

2. Difference Between Link vs NavLink :-



Feature	`Link`	`NavLink`
Basic Navigation	✅ Yes	✅ Yes
Active Styling	❌ No built-in support	✅ Adds automatic styling for the active route
Use Case	Simple navigation between pages	Navigation menus or tabs where you need to highlight the active link

3.Apply Active Class with NavLink

- You can use the className prop as a function to dynamically apply an active class.

Example:

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

function Nav() {
 return (
  <nav className="flex space-x-6 p-5 shadow-md">
   <NavLink
    to="/"
    className={({ isActive }) =>
     isActive ? "text-orange-500 font-bold" : "text-gray-800"
    }
   >
    Home
   </NavLink>

   <NavLink
    to="/about"
    className={({ isActive }) =>
     isActive ? "text-orange-500 font-bold" : "text-gray-800"
    }
   >
    About
   </NavLink>
  </nav>
 );
}

export default Nav;
```

isActive is a boolean that React Router passes to the className function — you can use it to conditionally apply your styles.

46. API in React

1. what is API ?

- Application Programming Interface.
- we need data form DB when making projects.
- But JS can not connect with Database.
- So we make API in other language as Java, PHP, or node etc.

Think of it like a waiter at a restaurant:

- You (the client) place your order (request)
- The waiter (API) takes it to the kitchen (server)
- The kitchen prepares your food (data)
- The waiter brings it back to you (response)

Example of API with JSON data :

```
{
  "location": {
    "name": "London",
    "country": "UK"
  },
  "current": {
    "temp_c": 16,
    "condition": {
      "text": "Partly cloudy"
    }
  }
}
```

47. Fetch Data from API

1. API methods :-

I) **GET** : To fetch data from a server
II) **POST** : To send new data to the server
III) **PUT** : To update existing data on the server
IV) **PATCH** : To partially update existing data
V) **DELETE** : To delete data from the server

React HandWritten Notes

2.Integrate API (React + fetch()) :-

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

const Api = () => {
  const [userData, setUserData] = useState([]);

  useEffect(() => {
    // Fetch the user data when component mounts
    getUserData();
  }, []);

  const getUserData = async () => {
    try {
      const url = "https://jsonplaceholder.typicode.com/users";
      const response = await fetch(url);
      const data = await response.json();
      setUserData(data);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };

  return (
    <div className="p-5">
      <h2 className="text-2xl font-bold mb-4">User List</h2>
      <ul>
        {userData.length > 0 ? (
          userData.map((user) => (
            <li key={user.id} className="mb-2">
              <strong>{user.name}</strong> ({user.email})
            </li>
          ))
        ) : (
          <p>Loading users...</p>
        )}
      </ul>
    </div>
  );
};

export default Api;
```

React HandWritten Notes