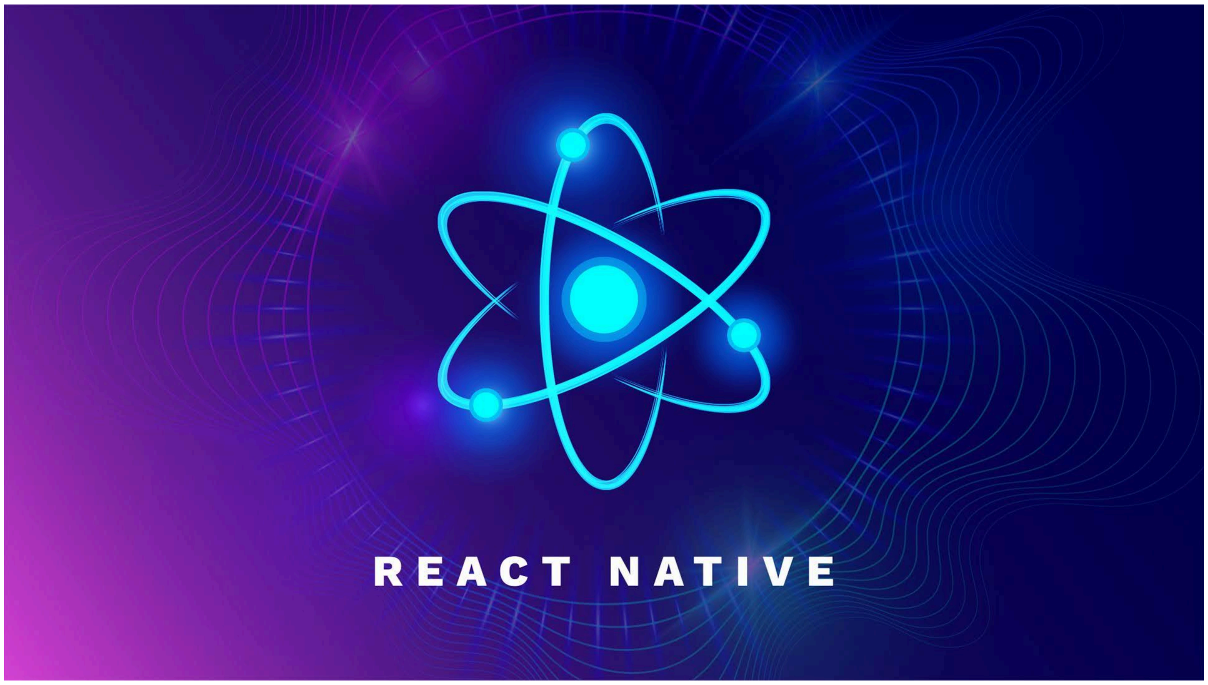

React

❚ (1). Introduction to React

- What is React?
  - Definition: A JavaScript library for building user interfaces.
  - Key Features:
    - Component-Based: Breaks the UI into reusable components.
    - Single-Page Applications (SPAs): Updates only parts of the page that have changed without reloading the entire page.
  - Example:
    On platforms like Instagram, as you scroll, new posts load without a full page refresh, making the experience fast and seamless.

❚ (2). Setting Up a React Environment

- Creating a React Project:

```
1- npx create-react-app project-name
2- cd project-name
3- npm start
```

❚ (3). JSX (JavaScript XML)

- Definition:
  An extension to JavaScript that allows you to write HTML-like code within JavaScript.
- Purpose:
  Simplifies the process of rendering UI in React components.
- Example:

```
const heading = <h1>Hello, React!</h1>;
```

▌(4). Components in React

- Definition:
  Components are small, reusable pieces of the UI that can be defined as JavaScript functions or classes.
- Benefit:
  They help break application into manageable & maintainable pieces.
- Types of Components:
  a) Functional Components
  b) Class Components

a) Functional Components ::

- Definition:
  JavaScript functions that return JSX.
- State Management:
  Use React Hooks (e.g., useState) to manage state and lifecycle.
- Lifecycle Methods
  No built-in lifecycle methods (hooks replace them).
- Example:

```
function Welcome() {return <h1>Welcome to React!</h1>;}
```

b) Class Components ::

- Definition:
  ES6 classes that extend React.Component.
- State Management:
  Use this.state for state and this.setState() for updates.
- Lifecycle Methods:
  Methods such as componentDidMount, componentDidUpdate.
- Example:

```
class Welcome extends React.Component {
       render() { return <h1>Welcome to React!</h1>; } }
```

▌(5). Props and State

a) Props:

- Definition:
  A JavaScript object used to pass data from a parent component to a child component
- Example:

```
// Child Component (receives props)
function Welcome(props) { return <h1>Hello, {props.name}!</h1>; }

// Parent Component (sends props)
function App() { return <Welcome name="Hasan" />; }
```

b) State:

- Definition:
  An object used to store dynamic data that can change over time

within a component

- When to Use:
  - • To track user input (e.g., form fields, buttons).
  - • To store data fetched from an API.
  - • To manage UI behaviors (e.g., modal visibility, toggles).
- Example:

```
function Counter() {
        const [count, setCount] = useState(0);
        return ( <button onClick={() => setCount(count + 1)}>
                    Clicked {count} times </button> ); }
```

▌(6). Component Lifecycle

a) Functional Components:

- Using React Hooks:

  The useEffect hook can mimic the behavior of several lifecycle methods (e.g., componentDidMount, componentDidUpdate, and componentWillUnmount).

b) Class Components:

- Phases:

  (1) Mounting Phase.
  (2) Updating Phase.
  (3) Unmounting Phase.

(1) Mounting Phase

- When: A component is created and inserted into the DOM.
- Key Methods:
  - • constructor(): Initializes state/props.
  - • render(): Returns JSX.
  - • componentDidMount(): Executes code after the component is added to the DOM (e.g., fetching data).

(2) Updating Phase

- When: State/props change or the parent component re-renders.
- Key Methods:
  - • shouldComponentUpdate(): Determines whether the component should re-render.
  - • componentDidUpdate(): Executes after the component has updated in the DOM.
  - • render(): Updates the UI.

(3) Unmounting Phase

- When: The component is removed from the DOM..

▌(7). React Hooks

- What are Hooks?:

  Functions that let you “hook into” React state and lifecycle features in functional components without writing a class.

- Hooks Types:
  (1) useState.
  (2) useEffect.
  (3) useContext.

(1) useState
- Syntax: const [stateVariable, setStateFunction] = useState(initialValue);
- Function:
  - Tracks and manages dynamic data within a component.
  - When the state changes, the component re-renders automatically.

(2) useEffect
- useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount.
- What are dependency ?:
  - in useEffect refers to a variable or state that the effect depends on.
  - If dependencies change, useEffect runs again.
  - If array empty ([]), useEffect runs once when component mounts.

- Use Cases:
  1️⃣ No Dependency Array → Runs on Every Render of Component
  - Syntax: useEffect(() => { console.log("Hi"); });

  2️⃣ Empty Dependency Array ([]) → Runs Only on Mount
  - Syntax: useEffect(() => { console.log("Hi"); }, []);

  3️⃣ Specific Dependencies → Runs When count Changes
  - Syntax: useEffect(() => { console.log("Hi:", count);}, [count]);

  4️⃣ Multiple Dependencies → Runs When count or name change.
  - Syntax: useEffect(() => { console.log("Hi:", count, name);
  }, [count, name]);

(3) useContext
- Why Use useContext ?

✅ Avoid "Prop Drilling" – No need to pass props manually on multiple levels.
✅ Global State – Share data (e.g., theme, auth state) across components.
✅ Easy to Use – Works with React.createContext().

- How useContext Works?

1️⃣ Import & Create a Context

```
import { createContext } from "react";
const ThemeContext = createContext("light"); // Default value: "light"
```

2️⃣ Provide the Context (Wrap Components)

```
import { useState } from "react";
import { ThemeContext } from "./ThemeContext";

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
export default ThemeProvider;
```

3️⃣Consume the Context in a Component

```
import { useContext } from "react";
import { ThemeContext } from "./ThemeContext";

const ThemeSwitcher = () => {
  const { theme, setTheme } = useContext(ThemeContext);

  return (
    <div>
      <p>Current Theme: {theme}</p>
      <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
        Toggle Theme
      </button>
    </div>
  );
};
```

▌ (8). Connecting React to the DOM (index.js)

- Definition
  A package that connects React with the browser's DOM.
- Role
  Used to render React components into HTML elements
- Entry Point
  The entry file that links your React app to the DOM.

▌ (9). Organizing the Component Tree (App.js)

- Definition
  A package that connects React with the browser's DOM.
- Role
  Acts as the root component of your React application.
- Responsibilities
  • Contains other child components.
  • Handles application-wide logic and routing.

▌ (10). Fragments & Component Instantiation

- Fragments (<> </>)
  • Used to group multiple elements without adding an extra node to the DOM.
  • Starting a Component (Rendering)

- Example :
  - (1) Without Parameters: <Greeting />
  - (2) With Parameters: <Greeting name={"Hasan"} />

▌(11). CSS Classes in React

- Definition

  In React, using className instead of class to assign CSS classes.
- Example:

  <h1 className="title">Hello, React!</h1>

(1) Inline Style (Object Syntax)

- Example: <h1 style={{ color: 'red', fontSize: '24px' }}>Hi</h1>

(2) Using a Style Object

- Example: const style = { color: 'red', fontSize: '20px' };

  <h1 style={style}>Hello, React!</h1>

▌(12). Spread/Rest Operator (...)

- Definition

  Works either to spread out elements or collect them together.
- Example:
  - Example in (1) Objects:
  - const arr = Object.keys(arr).map(key => ({ id: key, ...arr[key] }));
  - This useful if want to merge or add additional properties to object.

  - Example in (2) Arrays:
  - const combinedArray = [...arr1, ...arr2];
  - This is similar to using arr1.concat(arr2) but with cleaner syntax.

▌(13). Event Handling: e.target.value

- Definition

  Commonly used in event handlers (e.g., onChange) to access the value of an input field or form element.
- Example:

  const handleChange = (e) => { console.log(e.target.value); };
  <input type="text" onChange={handleChange} />

▌(14). Import & Export

- Importing
  - Same Folder: import Home from './home.js';
  - Different Folder: import Home from '../js/home.js';

- Exporting:

  (1) Default Export (Only one per file):
  - Export Syntax: export default home;
  - Import Syntax: import home from './home.js';

  (2) Named Export

- Export Syntax: export const home = () => { /* code*/ };
- Import Syntax: import { home } from './home.js';

● Note: Use default exports for the main component in a file.

▌(15). Events & Forms

● Common React Events
  • onClick
  • onChange
  • onSubmit
  • onKeyDown

● Handling Forms in React:
(1) Start the Form:
```
<form onSubmit={handleSubmit}>
    <button type="submit">Submit</button>
</form>
```
(2) Create the Handler:
```
const handleSubmit = (e) => { e.preventDefault(); };
```

● Note: Any variable used in forms should be managed via useState.

▌(16). Routing in React

● Overview
  • Used for navigation between pages in a Single Page Application.
  • Ensures the app does not reload the entire page when navigating.

● Installing
  • npm install react-router-dom

● Key Components

(1) <BrowserRouter>
  • Wraps the entire app to enable routing.
  • Should be placed at the top level (in index.js or App.js).
  • Never use multiple <BrowserRouter> in the full application.

(2) <Routes>
  • Holds multiple <Route> components and ensures only one page is shown at a time.
  • Example: <Routes>
  • <Route path="/" element={</>} />
  • <Route path="/" element={</>} />
  • </Routes>

(3) <Route>
  • Defines a mapping from a URL path to a component.
  • import { Routes, Route } from 'react-router-dom';
  • Example: <Route path="/" element={<Home />} />

(4) <Link>
  • Provides navigation without a full page refresh, replacing traditional <a> tags..
  • Example:

- import { Link } from 'react-router-dom';
- <Link to="/">Home</Link>

(5) <useNavigate>

- Allows programmatic navigation (e.g., after login, logout, or other actions)
- Example:
- import { useNavigate } from 'react-router-dom';
- const navigate = useNavigate();
- navigate("/dashboard");