ReactJS

Introduction

- ReactJS is an open-source JavaScript library for building **user interfaces**, especially for web applications.
- It was created by **Facebook (now Meta)** and is maintained by Facebook and a community of individual developers.
- It is used to create single-page applications (SPAs) with fast and dynamic UI updates.
- React focuses only on the view layer (the part users see and interact with).

History

2011 – Created by Jordan Walke, a software engineer at Facebook.

2013 – React was released as an open-source project.

2015 – React Native (for mobile apps) was introduced.

2020+ – Major improvements like Hooks, Concurrent Mode, and Server Components.

Features

- ✓ Component-Based Architecture
- ✓ Virtual DOM
- ✓ JSX (JavaScript XML)
- ✓ One-Way Data Flow
- ✓ Declarative Programming
- ✓ Reusable Components
- ✓ Fast Rendering

Why React Became Popular

- ✓ Component-based architecture
- ✓ Fast rendering with Virtual DOM
- ✓ Reusable UI components
- ✓ Strong Facebook backing
- ✓ Large community & ecosystem
- ✓ Works for web + mobile (React Native)

Core Concepts (Learn in this order)

1. JSX - JavaScript XML
2. Components (Functional vs Class)
3. Props - Passing data
4. State - useState Hook
5. Event Handling
6. Conditional Rendering
7. Lists & Keys

Installation & First App

1. Check node & npm version
2. Create reactapp using command

```
npx create-react-app myapp
cd myapp
npm start
```

```
npm create vite@latest myapp
cd myapp
npm run dev
```

How to Change Port

1. Open **vite.config.js** & Add

```
server:{
    port:4000
  }
```

1. Run: **npm run dev**

Cannot find module 'ajv/dist/compile/codegen'

1. Delete **node_modules** and **package-lock.json**
2. Reinstall dependencies: **npm install**
3. Try starting the app again: **npm start**

If It Still Doesn't Work

1. Manually install ajv package: **npm install ajv**
2. Try starting the app again: **npm start**

Dependency version mismatches or a corrupted node_modules folder

What is AJV?

- AJV stands for Another JSON Validator.
- AJV checks whether your JSON data (like API request/response or configuration files) is structured correctly and follows the rules you define.

Core Concepts

- ✓ let, const
- ✓ Arrow functions
- ✓ Array methods – map, filter, reduce
- ✓ Spread operator

Components

- Components are independent, reusable pieces of code that return HTML elements.
- Think of them as JavaScript functions that return HTML.
- A component is a reusable piece of UI in a React application.
- Like **LEGO blocks** - small pieces that you combine to build complex structures!

Types of Components

- ✓ **Functional Components** (modern & preferred)
- ✓ **Class Components** (older way, less common now)

1. Functional Components

- A simple JavaScript function that returns JSX.
- Must start with a capital letter.

Steps:

1. Create **Welcome.jsx** in src folder & **export** it
2. Import **Welcome.jsx** in **App.js**

Welcome.jsx

```
function Welcome(){
    return(
        <div>
            <h1>Welcome Page</h1>
        </div>
    );
}

export default Welcome;
```

App.jsx

```
import Welcome from './Welcome';

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

export default App;
```

2. Class Components

- Older React syntax using ES6 class.
- Rarely used now (Hooks replaced them), but good to know.

Steps:

1. Create **Welcome.jsx** in src folder & **export** it
2. Import **Welcome.jsx** in **App.js**

Welcome.jsx

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

class Welcome extends Component{
    render(){
        return(
            <div>
                <h1>Welcome Page</h1>
            </div>
        );
    };
}

export default Welcome;
```

App.jsx

```jsx
import Welcome from './Welcome';

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

export default App;
```

Props (Properties)

- Props (short for properties) are a way to send data from one component to another.
- From a parent component To child component.
- They make components dynamic and reusable.
- Props are read-only (you cannot change them inside the child component).
- Like passing ingredients to a recipe

Example – Single Value

Parent Component

```
import Childcomponent from "./Childcomponent";

function App() {
  return (
    <div>
      <Childcomponent name="Abcd"/>
    </div>
  );
}

export default App;
```

Child Component

```
const Childcomponent = (props) => {
  return (
    <div>Hello, {props.name} </div>
  )
}

export default Childcomponent
```

Example – Multiple Values

Parent Component

```
import Childcomponent from "./Childcomponent";

function App() {
  return (
    <div>
      <Childcomponent name="Abcd" age={20}/>
    </div>
  );
}

export default App;
```

Child Component

```
const Childcomponent = (props) => {
  return (
    <div>
        Hello, {props.name}<br/>
        Age, {props.age}
    </div>
  )
}
```

Destructuring Props (Cleaner Way)

```
const Childcomponent = ({name,age}) => {
  return (
    <div>
        Hello, {name}<br/>
        Age, {age}
    </div>
  )
}

export default Childcomponent
```

Different Types of Props

String Props

```
<Component text="Hello World" name="John" />
```

Number Props

```
<Component age={25} score={95.5} />
```

Boolean Props

```
<Component isStudent={true} hasGraduated={false} />
```

Array Props

```
<Component grades={[85, 90, 78]} subjects={["Math", "Science"]} />
```

Object Props

```
<Component student={{name: "John", age: 20}} />
```

Function Props

```
<Component onClick={handleClick} onUpdate={updateData} />
```

State

- State is an object that holds data or information about a component.
- state is managed inside the component (not passed from outside)
- When the state changes, the UI automatically re-renders to reflect those changes.
- It makes your React components interactive and dynamic.

React Hooks:

- ✓ useState
- ✓ useEffect
- ✓ useRef
- ✓ useContext
- ✓ useMemo
- ✓ useCallback

useState()

- React provides a built-in function called useState, called a Hook.
- We import it from "react" so we can use it inside our component.

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

- Create new component eg. **Counterapp.jsx**

Example 1

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

const Counterapp = () => {

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

  return (
    <div>
        <h2>Count: {count}</h2>
        <button onClick={()=>setCount(count+1)}>+</button>
        <button onClick={()=>setCount(count-1)}>-</button>
    </div>
  )
}
export default Counterapp
```

- count → current state value (variable)
- setCount → function to update the state

- Create new component eg. **Hideshow.jsx**

Example 2

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

const Hideshow = () => {
  const [show, setShow] = useState(false);
  return (
    <div>

        {show && <p>This is a secret message!</p>}

        <button onClick={()=>setShow(!show)}>
            {show ? "Hide Message" : "Show Message"}
        </button>
    </div>
  )
}
export default Hideshow
```

useEffect()

- useEffect is a React Hook that lets you run side effects in functional components.
- Side Effects include: Fetching API data, Running timers, Updating the page title, etc.

```
import {useEffect} from 'react'
```

```
useEffect(() => {
        document.title = `Count: ${count}`;
}, [count]);
```

useRef()

- useRef is a React Hook used to store a value that: Persists between re-renders, Does NOT trigger re-render when it changes.

- To access DOM elements (like document.getElementById)

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

const UseRefExample = () => {
    const inputRef = useRef();

    const handleClick = () => {
        alert("Value: " + inputRef.current.value);
    };

  return (
    <>
        <input ref={inputRef} type="text" />
        <button onClick={handleClick}>Show Value</button>
    </>
  )
}
export default UseRefExample
```

useContext()

- useContext is a React Hook that allows you to share data across multiple components without passing props.
- Without useContext → you must pass data from parent → child → grandchild → great-grandchild (called props drilling).
- With useContext → ANY component can access data directly.

UserContext.js

```
import { createContext } from "react";
export const UserContext = createContext();
```

App.jsx

```
import { UserContext } from './components/UserContext'
import Home from './components/Home'
import About from './components/About'

function App() {
const [user] = useState("Abcd");
  return (
    <>
      <UserContext.Provider value={user}>
        <Home />
        <About/>
      </UserContext.Provider>
    </>
  )
}
```

Home.jsx

```
import React,{ useContext } from 'react'
import { UserContext } from './UserContext'

const Home = () => {
    const name = useContext(UserContext);
  return (
    <div>Home {name}</div>
  )
}

export default Home
```

Lifecycle Methods

- constructor
- componentDidMount()
- componentDidUpdate()
- componentWillUnmount()

useEffect

```
// Equivalent of componentDidMount: runs only once after initial render
 useEffect(() => {
  console.log("Component did mount");

  // Equivalent of componentWillUnmount: cleanup function
  return () => {
   console.log("Component will unmount");
  };
 }, []); // Empty dependency array ensures it runs only once
```

➢ useEffect hook syntax

```
useEffect(() =>{
// side effect code
     return()=>{
        // clean up code (optional)
      };
},[dependancies]);
```

➢ Parameters

- ➢ Dependencies Array.
- ➢ Run every render: Omit the array
- ➢ Run once on mount: Use []
- ➢ Run on variable changes: [someVar]

useEffect

- What is Side Effects.
- React just have the pure components.
- Data which is used which declared inside the function
- For the sideEffect some functions get the data from the outside of the function.

```
// Example side effect starts
    // let a =10; // API
    // function display(){
    //     a+=1; //
    // }
    // display();
    // console.log(a);
// Example side effect Ends
```

useEffect

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

const Counterapp = () => {

const [count, setCount] = useState(0);
 useEffect(()=>{
        // Code for side effect
        console.log("count",count);
        setInterval(() => {
            const updateDate = new Date();
            setDate(updateDate.toLocaleTimeString());
        }, 1000);
        // return()=>{
        // }
    },[count])
```

```
// Equivalent of componentDidUpdate:
 runs every time 'count' changes (excluding first mount)
  useEffect(() => {
    if (count > 0) {
      console.log("Component did update - count changed");
    }
  }, [count]); // Dependency array watches 'count'


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

useref

- useRef is a React Hook that lets you reference a value that's not needed for rendering.

- **useRef syntax**

```
const ref = useRef(initialValue);
```

useref

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

function Userefcomponent() {
   const username = useRef(null);
   const password = useRef(null);

   function handleSubmit(e){
      e.preventDefault();
      console.log(username.current.value);
      console.log(password.current.value);
   }
```

useContext

What is Context in React?

- Context is a built-in feature in React that allows you to share data across multiple components without having to pass props manually through every level of the component tree.

- If you pass props manually from parent → child → grandchild, it becomes **prop drilling**.

Example

1. Create Home component
2. Create Profile component
3. Create UserContext for create context & create provider
4. Use UserContext / provider in app.js

Example: UserContext.jsx

```
import React, { createContext, useState } from "react";

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

```
// Step 2: Create Provider component
export function UserProvider({ children }) {
  const [user, setUser] = useState(null);

  const login = (name) => setUser(name);
  const logout = () => setUser(null);

  return (
    <UserContext.Provider value={{ user, login, logout }}>
      {children}
    </UserContext.Provider>
  );
}
```

Example: Home.jsx

```
import React, { useContext, useState } from "react";
import { UserContext } from "./UserContext";

export default function Home() {
  const { user, login, logout } = useContext(UserContext);
  const [name, setName] = useState("");

  return (
    <div>
      ........
    </div>
  );
}
```

```
{user ? (
<>
  <h3>Welcome, {user}!</h3>
  <button onClick={logout}>Logout</button>
</>
) : (
<>
  <input type="text“
  placeholder=“Your name” onChange={(e) => setName(e.target.value)}/>

  <button onClick={() => login(name)}>Login</button>
</>
)}
```

Example: Profile.jsx

```
import React, { useContext } from "react";
import { UserContext } from "./UserContext";

export default function Profile() {
  const { user } = useContext(UserContext);

  return (
    <div>
      <h3>Profile Page</h3>
      {user ? <p>Logged in as {user}</p> : <p>No user logged in.</p>}
    </div>
  );
}
```

Example: App.jsx

```
import { UserProvider } from "./components/UserContext";
import Home from "./components/Home";
import Profile from "./components/Profile";


function App() {
  return (
    <UserProvider>
    <div>
      <Home />
      <Profile />
    </div>
    </UserProvider>
  );
}
export default App;
```

useContext

```
// CONSUMER COMPONENTS
// 1. import React, { useContext } from 'react';
// import { MyContext } from './ComponentA';
// 2. const value = useContext(MyContext);
```

useMemo

- Higher-order component that prevents unnecessary re-renders.
- Improves the performance by caching the result of components render
- When props don’t change, React skips the re-render.
- Use shallow comparison to check whether the props have changed