**React ?**

React is a **JavaScript library** built by **Facebook** that is used to create **User Interface (UI)** and **Single Page Applications (SPAs).**

**SPA ?**

A **Single Page Application (SPA)** is a web application that loads in a single html webpage and dynamically loads and update the content on the page without reloading it.

**React flow :**

App.jsx 🡪 main.jsx 🡪 index.html 🡪 website/browser

**node\_module**

This package is automatically created when we run **npm install**. It contains all the dependencies required for React.

**package.json**

It is a configuration file in NodeJs Project. This file includes in React to store the **metadata** and **dependencies** of React application.

**package-lock.json**

This file is **automatically generated** file that records the **exact version of installed dependencies**

**Features of React :**

1. **Virtual DOM (Document Object Model) :**

The Virtual DOM is a lightweight copy of Real DOM. It is used to optimize rendering performance and efficient UI updates.

Instead of directly modifying Real DOM React first updates the VDOM, calculate difference **(diffing process)** form old VDOM to VDOM and then update the necessary changes to the Real DOM **(Reconciliation).**

**Diffing :**

It is an algorithm in React uses during Reconciliation to find the difference between old VDOM and the new VDOM (pre-updated vdom)

1. **JSX :**

**JSX (JavaScript XML)** is a **syntax extension** for JavaScript used in React to define User Interface (UI) Structure using JavaScript file. React uses **babel compiler** to convert modern jsx into plain JavaScript

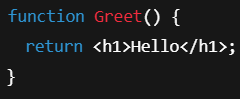
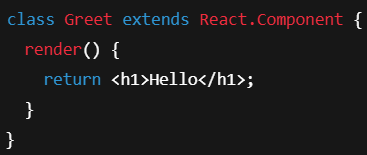
1. **Component Based Architecture**

Components are **reusable** and **independent block of code** which are used to describe the user interface. First react component App.jsx. It returns jsx single HTML element. Component name always start with uppercase.

Types – functional comp, class-based comp

**Difference Between Functional and Class-based Component?**

| **Aspect** | **Functional Component** | **Class Component** |
| --- | --- | --- |
| **Syntax** | Plain JavaScript function | ES6 class that extends React.Component |
| **State** | Uses useState hook to manage state (since React 16.8) | Uses this.state and this.setState() |
| **Lifecycle** | Uses hooks like useEffect | Uses lifecycle methods like componentDidMount, componentDidUpdate |
| **Code Simplicity** | Simpler, more concise | More verbose |
| **Performance** | Slightly faster due to being just functions | Slightly heavier due to class overhead |
| **‘this’ keyword** | Not used | Must use this to access props, state, and methods |

**Lifecycle of React Component**

The components lifecycle refers to the sequence of methods invoked( active or executed ) at different stages of a component’s existence.

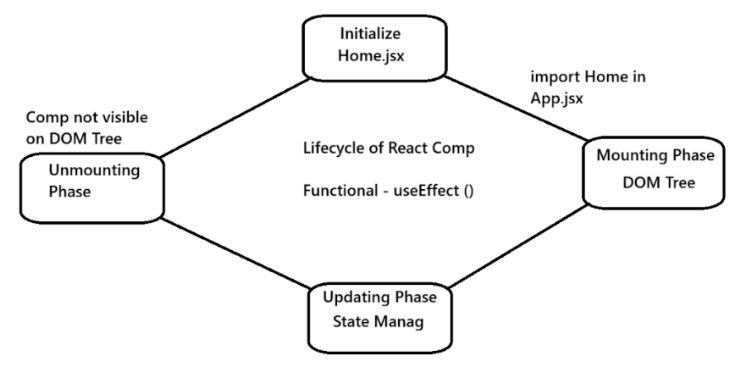
Main three phases are – Mounting Phase, Updating Phase, Unmounting Phase.

   Classbased       Functional   Phase

componentDidMount() **useEffect(()=>{  }, [ ])             M**

componentDidUpdate() useEffect((**)=>{ },[count ]**)                 U

componentwillUnmount() useEffect(**()=>{ return ()=>{....}}, [ ]**) Unm

****

**How to iterate Array of objects?**

Use map()

arrayName.map(() => { return jsx })

**What is props and state?**

**Props:**

Props stand for **properties**. It is used to **transfer data** from parent comp to child comp. Props are **immutable** (read-only)

**State :**

**State refers to a built-in Object which is used to store property values of component.** State is an updatable structure which can be update on user action.

React re-render the components to reflect the new state

This state managed within the component

1. Functional based comp - useState()
2. Classbased comp - this.state

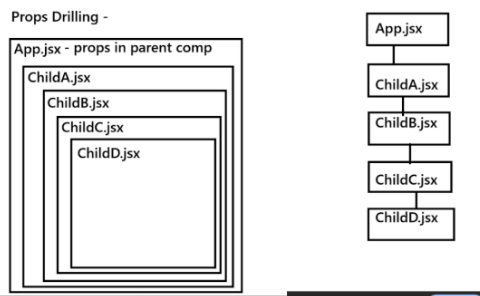
State is mutable, meaning it can be change and change will cause the comp re-render

**What is StrictMode ?**

1. StrictMode in React is a tool for highlighting potential problems in a react application.
2. It doesn't render anything visible to the user - it is only used during the development phase and helps you to write better react code by identifying unsafe lifecycle methods.
3. Always present in main.jsx

**Props Drilling ?**

Props drilling occurs when data needs to be pass through multiple layers of comp to reach a deeply nested component. To handle the props drilling React uses Context API concept.



**Context API -**

1. The Context API is a feature in React that allows you to share data across components without having to pass props manually at every entry level
2. It helps in managing global state efficiently, avoiding “props drilling”

**Key Comp of Context API -**

1. **createContext()** - Create a context object
2. **Provider** - Supplies the data to consume comp
3. **Consumer** - Used to Consume the context value
4. **useContext ()** Hook

**Events**

1. **onChange Event:**

It is used to detect the changes on the input field (text, check boxes, select menus etc). It gets triggered every time the input value changes.

**event.target.value:**

It is used to access the current value of the input element that triggered the event.

* Event – an event is an object
* Target – it refers to the element on which event occurred.
* Value – it gives the updated value of that element

**Hooks in React:**

**Hooks are functions that make Functional components work like Class components.**

1. A hook is a special function in React that allows you to use state and other React Features in functional components.
2. Hooks were introduced in React 16.8 version to replace class components
3. Hooks has its own syntax
4. Hooks are always imported top of the component

**useContext() Hook :**

**useContext** is used to **manage global data** in react application**. E.g. global state, services, themes, user settings.**

The useContext Hook access the value of React Context directly in functional comp without needing a Context.Consumer wrapper (we never use - it creates callback hell situation ).

Syntax –

**const contextValue = useContext(contextObject)**

**create , provide, use**

**useState() Hook :**

useState Hook is a functional to **add State** in Functional Component. A State is nothing but just **values or variables** of your component.

This is a fundamental hook in react which is used to manage the state of react components in functional.

useState() hook return Array-destructuring format

**Syntax –**

**const [state, setState] = useState(initialValue)**

state - hold the initial/current state of react comp

setState - This is the function used to update the state

initialValue - This is starting value of the state (number, string, object, array etc)

**useEffect() Hook :**

useEffect is used to **perform side effects** in our component. Side effects are **actions** which are performed with the “Outside world”. We perform a side effect when we need to reach outside of our React components to do something. E.g. Fetching data from API, updating the DOM document & window, Timer functions – setTimeout & setInterval.

Syntax - **useEffect( callback, dependencies )**

callback – what to run

dependencies – when to run

**Class Component** – componentDidMount, componentDidUpdate, componentWIllUnmount

Variation of useEffect

* useEffect without dependencies – it runs with first render and also run on any thing changes
* useEffect with empty array – it runs only on first render.
* useEffect with variable – it runs on first render and runs with that variable.

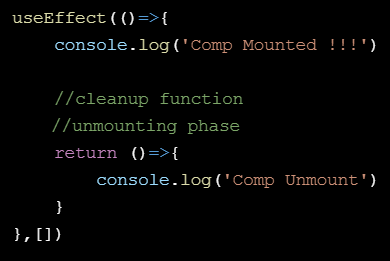
 Classbased       Functional   Phase

componentDidMount() **useEffect(()=>{  }, [ ])             M**

componentDidUpdate() useEffect((**)=>{ },[count ]**)                 U

componentwillUnmount() useEffect(**()=>{ return ()=>{....}}, [ ]**) Unm

**clean up function**

****

**useRef() Hook**

useRef allows us to **access DOM elements**. It is used for **creating mutable variables** which will **not re-render the component.**

**Syntax – const variableName = useRef(initialValue)**

**useReducer() Hook**

useReducer is used to **manage state** in our react application**.** In other words, useReducer **works like state management tool.**

**State Management** is used to **manage all states** of application in a simple way. Always use useReducer Hook **when you have a lot of states** **and methods** to handle.

**Syntax - const [state, dispatch] = useReducer(reducer, initialState)**

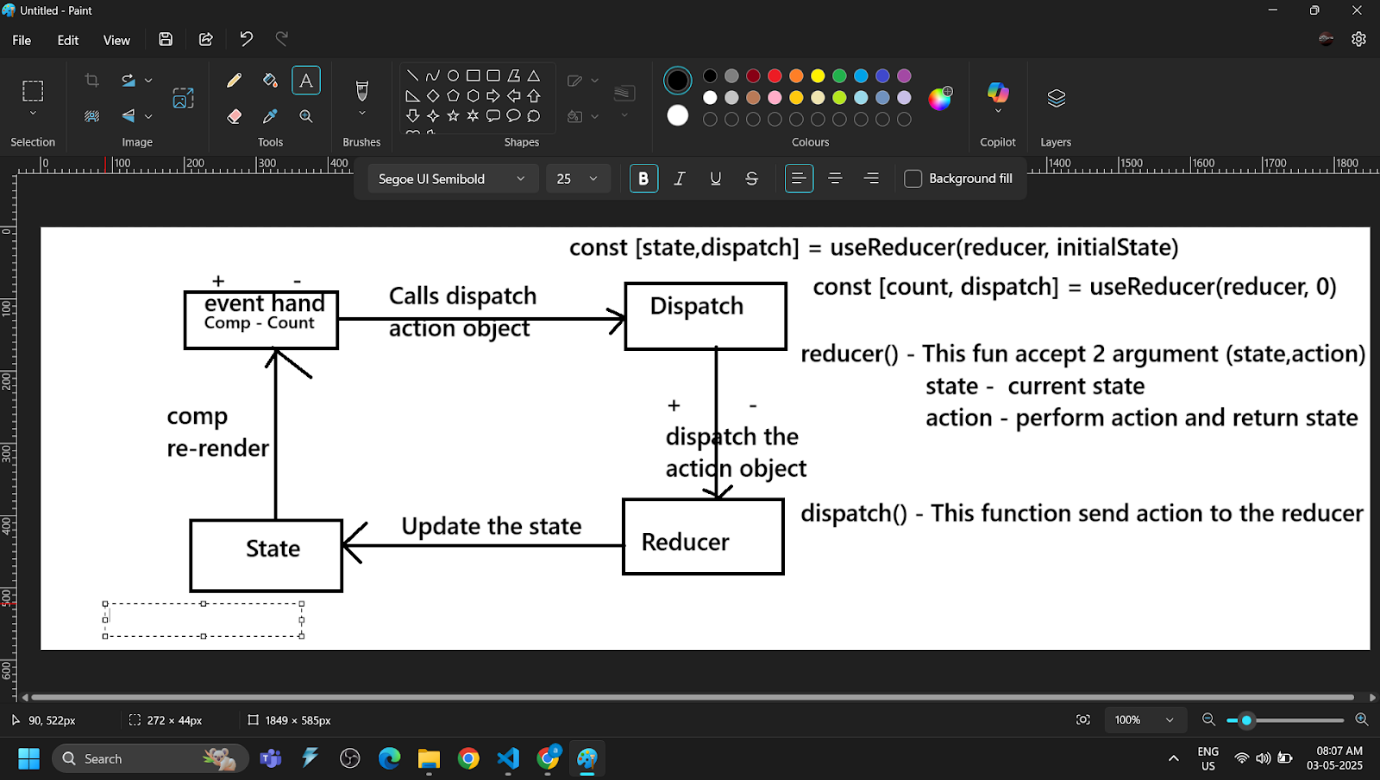
**Reducer –** A pure function that takes two arguments the current state and an action and returns the new state.

**Action Object –** describes what to do it usually has a type property and optionally a payload**.**

**State Managemant –** perfect for scenarios where a state has multiple sub-values or depends on the previous state.

**Properties**

* Reducer – A function that defines how the state should update based on an action.
* initialState – the starting value of the state
* state – the current state
* dispatch – a function to send actions to the reducer.

****

**Payload –** it refers to the actual data that is sent with an action or request. It is a part of the action object sent to the reducer. Typically contains data needed by the reducer to perform its operation**.**

| **Hook** | **Use it when you need to...** |
| --- | --- |
| useState | Store and update local component state (like form data, toggles, counters). |
| useEffect | Run side effects (fetch data, set up subscriptions, manipulate DOM, etc.) after render. |
| useRef | Reference DOM elements (like <input>) or persist mutable values across renders **without re-rendering**. |
| useContext | Share values (like theme, auth, language) between components without **props drilling**. |
| useReducer | Manage **complex state logic**, especially when state depends on previous state (e.g., like Redux). |

**I want to store my data and it should not vanish if I refresh my page ?**

Use **local storage**

It is a feature provided by web browser that allow web application to store data in key-value pair. When we close or reopen the application the data remains persist. E.g. Shopping cart, User setting etc.

Data does not expire unless manually removed it can store up to 10mb data. Key must be in String format. Data is only accessible from the same domain that is stored.

Methods

localStorage.setItem(key, value)

localStorage.getItem(key)

localStorage.removeItem(key)

localStorage.clear()

JSON.stringify() – local storage only store key in the string format so to store object or array we must convert them into string format we use **JSON.stringify().**

JSON.parse() – To retrieve the converted object into original form then use JSON.parse()