**What is Library?**

A collection of pre-written code that programmers can use to perform specific tasks or add functionality to a program. Libraries are often written in a specific programming language and are designed to be reusable**.**

**What is React?**

React is an **open-source front-end JavaScript library** also called as **component based** front end library that is used for building user interfaces, especially for single-page applications. It is used for handling **VIEW LAYER** for web and mobile applications so that its light weight application.

React helps you manage these components efficiently and keeps your UI in sync with your data.

React was created by [Jordan Walke] (https://github.com/jordwalke), a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

Whenever you open ecommerce website The request will hot the UI server. This server will have all static content for dynamic content it will send API server it will response for fetching the data from data server.

**What are the major features of React or Purpose of React?**

The major features of React are:

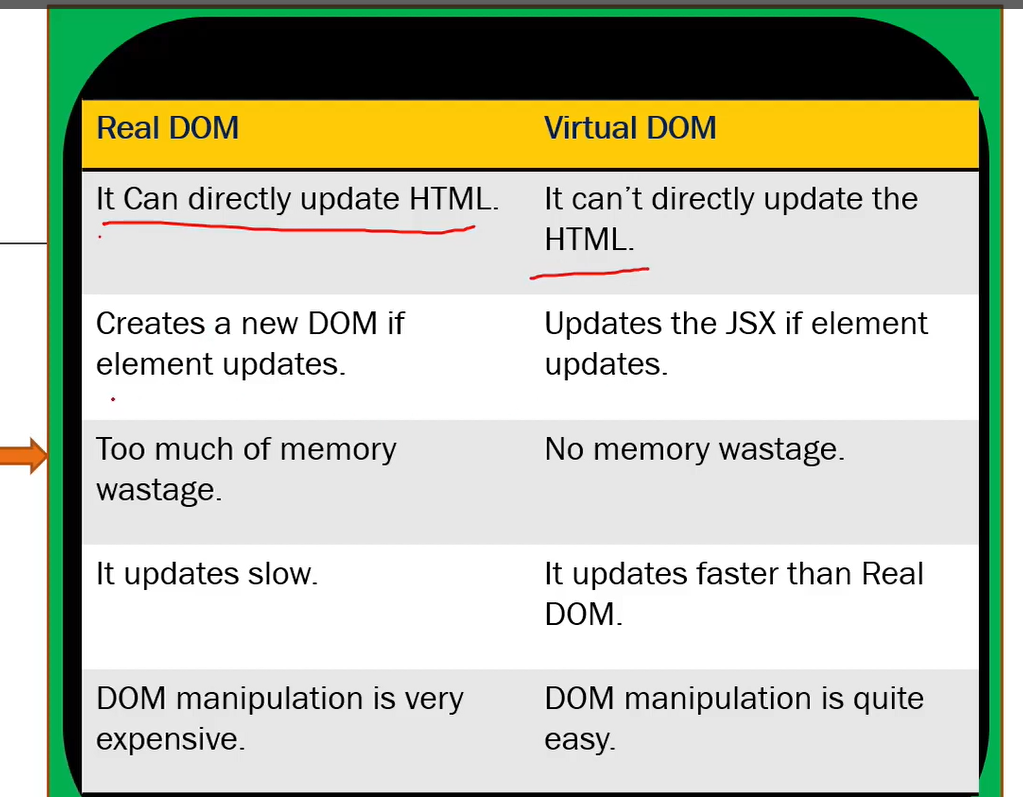
* It uses **VirtualDOM** instead of RealDOM considering that RealDOM manipulations are expensive.

**what is DOM?**

Document object model which represents the web page as a treelike structure which allows JavaScript to dynamically access and manipulate the content and structure of web page, we can achieve everything using Real DOM but the performance will be low.

**what is virtual DOM.**

Copy of real DOM will be saved in Virtual DOM, whenever the change happens in any of the component that particular change will be compared and that alone will be re render, other words virtual representation of a user interface is preserved in memory and synchronized with the browser's DOM minimizing direct manipulation of the actual DOM so that performance will not affect much.



* Supports **server-side rendering**.
* Follows **Unidirectional** data flow or data binding.
* Uses **reusable/composable** UI components to develop the view(maintaining and scalable approach)

**what is Component.?**

Web consists of many parts; each part is a component

Each components has states (interface changes) and props are objects (sharing the access with the components, its immutable we can get the data but unable to modify).

**Advantages of React.?**

* Used to build sing page Application (SPA) by using components. (React follows component-based architecture which allows developer to use reusable components which can be used throughout the application and that makes its heavy speed.
* React.js is cross flatform and open-source JavaScript framework library, free to use. React is used to develop mobile and applications.
* Light weight and very fast virtual DOM. (Very fast due to Virtual dom)
* Large Community and Ecosystem.
* Testing is easy.

**What is SINGLE PAGE APPLICATION.?**

The web page which consists many components, if there is any change/ update in any component, only that content in the component is dynamically updated instead of refreshing the whole page.

**What is Component ?**

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

**In how many ways you can create Component ?**

Components come in two types, Class components and Function components

**What is state and How to Update State ?**

**state** is an object that holds or allows you to store property values that belong to the component that may change over the lifetime of a component. It is essentially the data that drives the behaviour and appearance of your component. When the state of a component changes, React re-renders the component to reflect the updated state.

- Each component can have its own state.

- The state is **mutable**, meaning it can change over time, usually in response to user actions or network responses.

- State is used in **class components as this.state** and in **functional components using useState** hook.

Updating State in Functional Components

- The **useState** hook returns an array with two elements: the current state value and a function to update it.

**- Example:**

import { useState } from "react";

function Counter() {

  const [Count, SetCount] = useState(0);

  return (

    <>

       <h1> Performing calculations:: {Count}</h1>

       <button onClick = {() =>SetCount(Count+1) }> Adding </button>

       <button onClick = {() =>SetCount(Count-1)}> Subtracting </button>

        <button onClick = {() =>SetCount(Count\*2)}>Multiplication    </button>

    </>

  );

}

export default Counter;

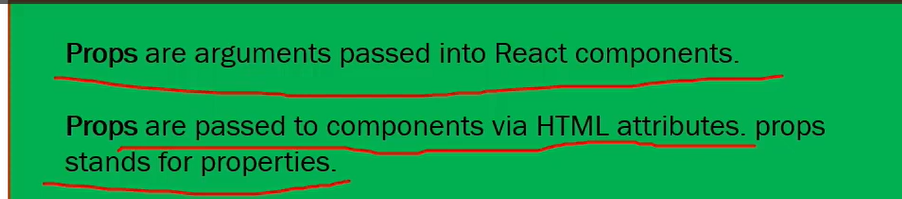
**- Here, `count` is the current state, and `setCount` is the function to update the state.**

State management is a core concept in React, enabling you to create interactive and dynamic user interfaces.

**What happens when setState() is called ?**

React initiates an efficient, batched update process that eventually leads to the component being re-rendered with the updated state, ensuring that the UI stays in sync with the underlying data.

**What is Props ?**



**Why we use Props ?**

Props allow you to pass data and behavior (via functions) from a parent component to a child component.

function Component1 (){

const [user] = useState('A')

return(

<>

<h1> Child1 component</h1>

<Child1 user = {user}/>

</>

)

}

function Child1 ({user}){

return(

<>

<h1> Child2 component</h1>

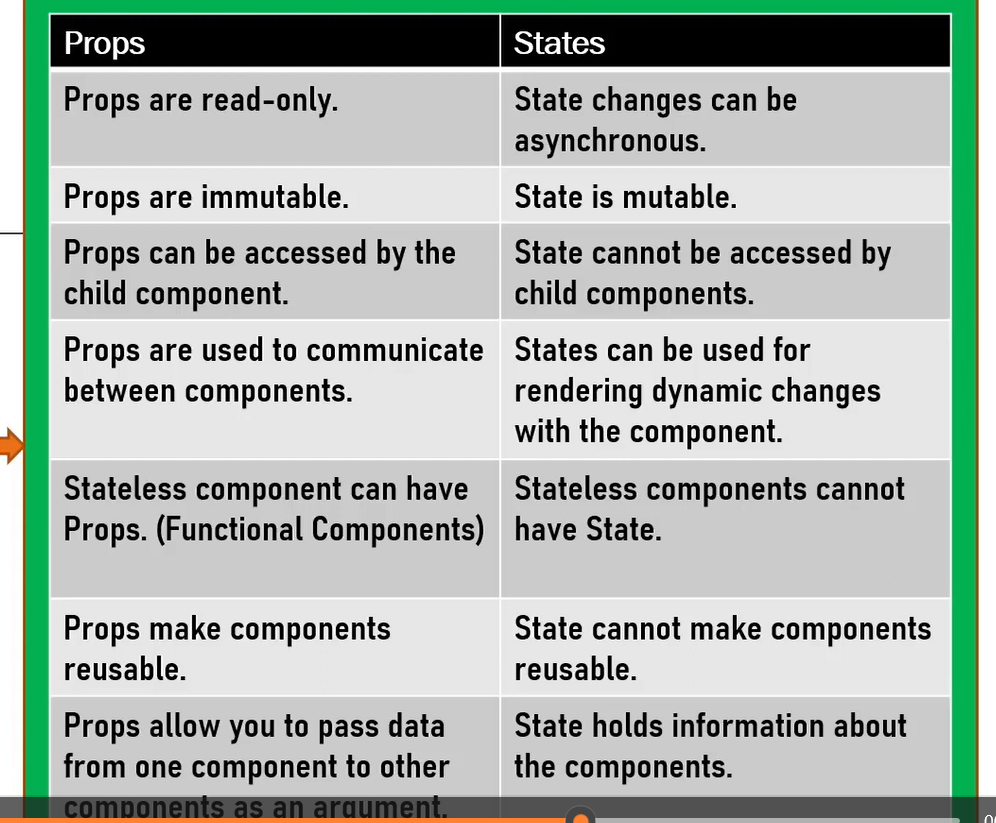
<Child2 user= {user}/>

</>

)

}

**What is difference between props and states?**



**How to create Parent Child Relation ?**

Creating a parent-child relationship in React involves nesting components, where one component (the parent) renders another component (the child) and often passes data or functions as props.

**What is Application Flow ?**

App Initialization: The React app starts, and the root component (<App />) is rendered.

User Logs In: The user inputs credentials into a form and submits it.

State Update: The login form triggers an API call to validate the user. Upon success, the user’s data is stored in the global state.

Navigation: The user is redirected to the dashboard, where the relevant data is displayed.

Data Fetching: The dashboard component fetches additional data (e.g., user’s recent activity) and updates its state accordingly.

User Interaction: The user interacts with various elements on the dashboard, triggering updates in the UI and possibly making further API calls.

Logout: The user logs out, triggering a cleanup of user data from the state and redirecting back to the login page.

**What is diff between render() in React and render() in ReactDOM ?**

**render() in React**

* The render() method is a lifecycle method in class components that defines what the component's UI should look like.
* Purpose: It returns the JSX that represents the component's structure. This JSX is then converted to actual DOM elements by React.
* render() function from the ReactDOM library is used to render a React element (or component) into a specified DOM container on the page.

**render() in ReactDOM ?**

* Purpose: It serves as the entry point to rendering a React application or component tree into the DOM.

**In how many ways we can pass the data from parent to child ?**

* Props: The most straightforward and common way to pass data.
* Callback Functions: Allows the child to communicate back to the parent.
* Context API: Useful for avoiding prop drilling in deeply nested components.
* State Management Libraries: Ideal for complex, large-scale applications.
* Render Props: A pattern for passing data in more dynamic scenarios.
* Higher-Order Components (HOCs): A pattern for sharing logic between components.
* Composition: Passing JSX or components through the children prop for more flexible component design.

**In how many ways you can manage state in React Applicatiion ?**

* Local State (useState, setState): Best for managing state within individual components.
* Derived State (useMemo, useCallback): Useful for optimizing performance by memoizing expensive calculations or functions.
* Lifting State Up: Essential for sharing state between sibling components.
* Context API: Ideal for sharing state across a component tree without prop drilling.
* Redux: A powerful, widely-used global state management solution, especially in large applications.

**What is Context Api and Why do we use Context API ?**

* Context API is a way to pass data through the component tree without having to pass props down manually at every level.
* Use Case: It's primarily used to avoid prop drilling and to manage global state that needs to be accessed by multiple components.
* Key Components: createContext, Provider, Consumer (or useContext hook).
* Advantages: Simplifies state management for certain use cases, reduces dependency on parent components, and provides a more maintainable and scalable solution for passing data across components.

**Which kind problem will be solved using Context API ?**

The Context API is a powerful tool for solving various problems related to state management and data sharing in React applications, including:

* Avoiding prop drilling.
* Managing global state.
* Decoupling components.
* Sharing configuration or constants.
* Handling user preferences or localization.

**What is axios ?**

Axios is a powerful and flexible library for handling HTTP requests in JavaScript applications. Its features, such as promise-based handling, support for various HTTP methods, interceptors, and built-in error handling, make it a popular choice for making network requests and integrating with APIs in both browser and Node.js environments

**Anything alternate to axios ?**

Fetch API: Native, modern, and flexible; built into browsers.

jQuery AJAX: Part of jQuery; suitable if you're already using jQuery.

Superagent: Lightweight and flexible with a simple API.

Got: Advanced features for Node.js environments.

Ky: A small and elegant wrapper around Fetch.

node-fetch: Provides Fetch functionality for Node.js.

Apollo Client, Relay: Specialized for GraphQL queries and state management.

**How to implement the routing the react application ?**

React Router is the most commonly used library for routing in React applications.

Basic Setup: Install React Router, create components, and define routes using BrowserRouter, Routes, and Route.

Navigation: Use Link instead of traditional <a> tags for navigation.

Nested Routes: Define nested routes using Outlet.

404 Handling: Add a fallback route for handling non-existent paths.

**What is Redux ?**

**Why we have to use redux ?**

**What is Reducer ?**

**How redux store is available to react app ?**

**What is redux Logger ?**

**How to call the reducer ?**

**Can you explain Redux flow ? ()**

Redux is a state management library commonly used with React to manage the state of applications in a predictable way. The Redux flow refers to the sequence of steps and interactions involved in managing state changes within a Redux-based application.

**Action Creation**: An action is created (either directly or using an action creator).

**Dispatching Action**: The action is dispatched to the store.

**Reducer Handling**: The store sends the action to the appropriate reducer.

**State Update**: The reducer processes the action and returns a new state.

**State Change Notification**: The store updates its state and notifies subscribed components or listeners.

**Component Rendering**: Components access the updated state (usually via mapStateToProps or hooks like useSelector) and re-render based on the new state.

**What are hooks ?**

Simplify state management in functional components with useState.

Perform side effects with useEffect.

Access context values with useContext.

Manage complex state logic with useReducer.

Optimize performance with useMemo and useCallback.

Access and persist values or DOM elements with useRef.

**Why hooks were introduced?**

**Hooks were introduced to:**

* Simplify state management and side effects in functional components.
* Facilitate the reuse of stateful logic through custom hooks.
* Improve the readability and maintainability of components.
* Align with functional programming principles.
* Avoid issues associated with complex component hierarchies and wrapper hell.

**What is useEffect or How useEffect will maintain 3 lifecycle method ?**

* useEffect provides a way to handle side effects in functional components.
* Replicates componentDidMount: Run useEffect with an empty dependency array.
* Replicates componentDidUpdate: Run useEffect with dependencies to react to changes.
* Replicates componentWillUnmount: Return a cleanup function from useEffect to handle cleanup logic.

**What is the use of useContext() ?**

The useContext() hook in React is used to access the value of a context in functional components. Context is a way to pass data through the component tree without having to pass props down manually at every level.

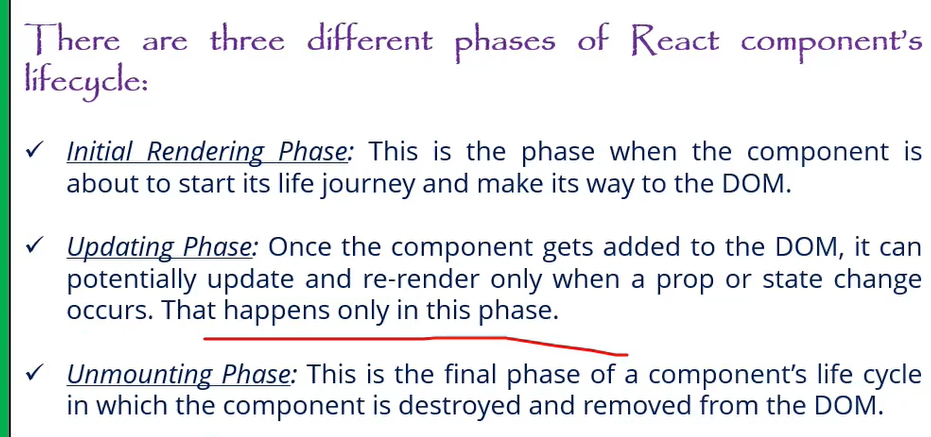
**how can you create reference for element in functional Component ?**

**What is the diff btw useState and useReducer**

**How to use Redux in Functional Component ?**

**What is the diff between functional and Class components.**

|  |  |
| --- | --- |
| **Functional component** | **Class component** |
| No need to import react. | Import of (react & component) must include |
| Normal function syntax with return () is must. | **Class App extends Component** is must, in order to act as react component |
| light weight | Extra structure is needed. |
| Destructing will be done by passing within args | Extra line needed for destructuring, Method and variables |
| Here we cannot use life cycle methods in functional components we can handle it using HOOKS except **ComponentDidCatch** error boundaries. (higher level error catch). | Render () is also react lifecycle method. |
|  |
| Mounting- **ComponentDidMount** |
| Updating- **ComponentDidUpdate** |
| Unmounting- **ComponentWillMount** |
| function Heva (){  return (  <>  <p>She is playing</p>  <p>Here hobby is playing</p>  </>  ) } | Class component: Methods and   import React, { Component } from 'react';  class classComponent extends Component {  render() {  return (  <div>    </div>  );  } }  export default class; |



**"Can you explain the main differences between React class components and functional components with hooks? Why might you choose one over the other?"**

**Lifecycle Methods in Class Components:**

- \*\*Mounting:\*\* - `componentDidMount`: Called once the component is mounted.

- \*\*Updating:\*\* - `componentDidUpdate`: Called after the component updates due to changes in props or state.

- \*\*Unmounting:\*\* - `componentWillUnmount`: Called right before the component is unmounted and destroyed.

**Note**: `componentWillMount` is not recommended for use anymore due to potential issues and is considered legacy. For mounting, use `componentDidMount` instead.

2. \*\*Functional Components with Hooks:\*\*

- Functional components do not have lifecycle methods directly. Instead, you use hooks to achieve similar functionality:

- \*\*`useEffect` Hook\*\*: This hook can replicate `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` behavior. By specifying dependencies, you can control when effects run.

- \*\*`useState` Hook\*\*: Used for managing state in functional components, replacing the need for `this.state` and `this.setState`.

\*\*Why Choose One Over the Other?\*\*

- \*\*Functional Components\*\*: Generally preferred for their simplicity and ease of use. Hooks make it easier to share logic and manage state without worrying about the complexity of class components.

- \*\*Class Components\*\*: Might be used for legacy codebases or when you need specific lifecycle methods not yet covered by hooks (although hooks generally cover all use cases).

Would you like to elaborate more on this or move to the next question?

**What is JSX?**

\*JSX\* is a XML-like syntax extension to ECMAScript \*JavaScript XML\*. primarily used with React to describe what the UI should look like. JSX allows you to write HTML-like code within JavaScript, making it easier to create and manipulate UI components, giving us **expressiveness of JavaScript** along with HTML like template syntax.

JSX converts to JavaScript with help of babel transpiler, because browser understands only javascript.

Basically it just provides syntactic sugar for the `React.createElement()` function.In the example below text inside `<h1>` tag is returned as JavaScript function to the render function.

function MyFirstComponent() {

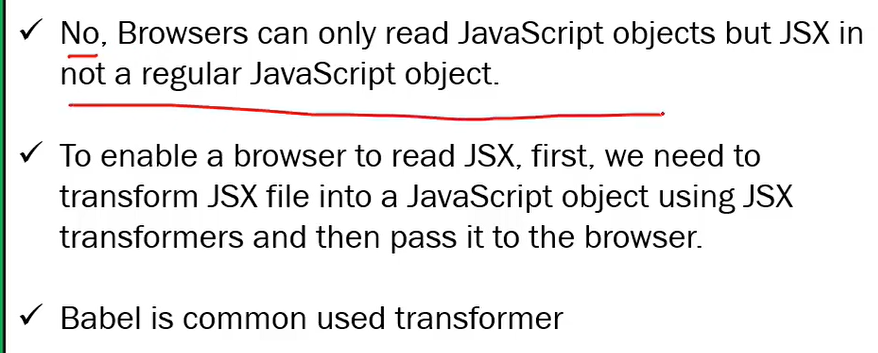
return (

< >

 <h1> My First React Component! </h1>

</ >

  );}



**what are the main files in React.js**

Minimum 6 filers are.

* Index.html file-🡪 Though single page application which is under public folder.
* Components🡪 The components which should be in application are used to place under src folder only.
* App.js🡪 Main component. or root components
* App.test.js-🡪 unit test
* Index.css-🡪 glogal css file.
* Index.js file🡪 entry point of javascript file

**Why to use React over other framework**

• Performance: React excels at handling dynamic UIs with

minimal re-renders, keeping your app fast and snappy.

• Composability: React components promote code reusability,

making development faster and easier to maintain.

• JSX: JSX syntax blends JavaScript with HTML, making UI

creation intuitive for developers.

Large Community: A vast community and ecosystem provide support,

libraries, and tools for rapid development.

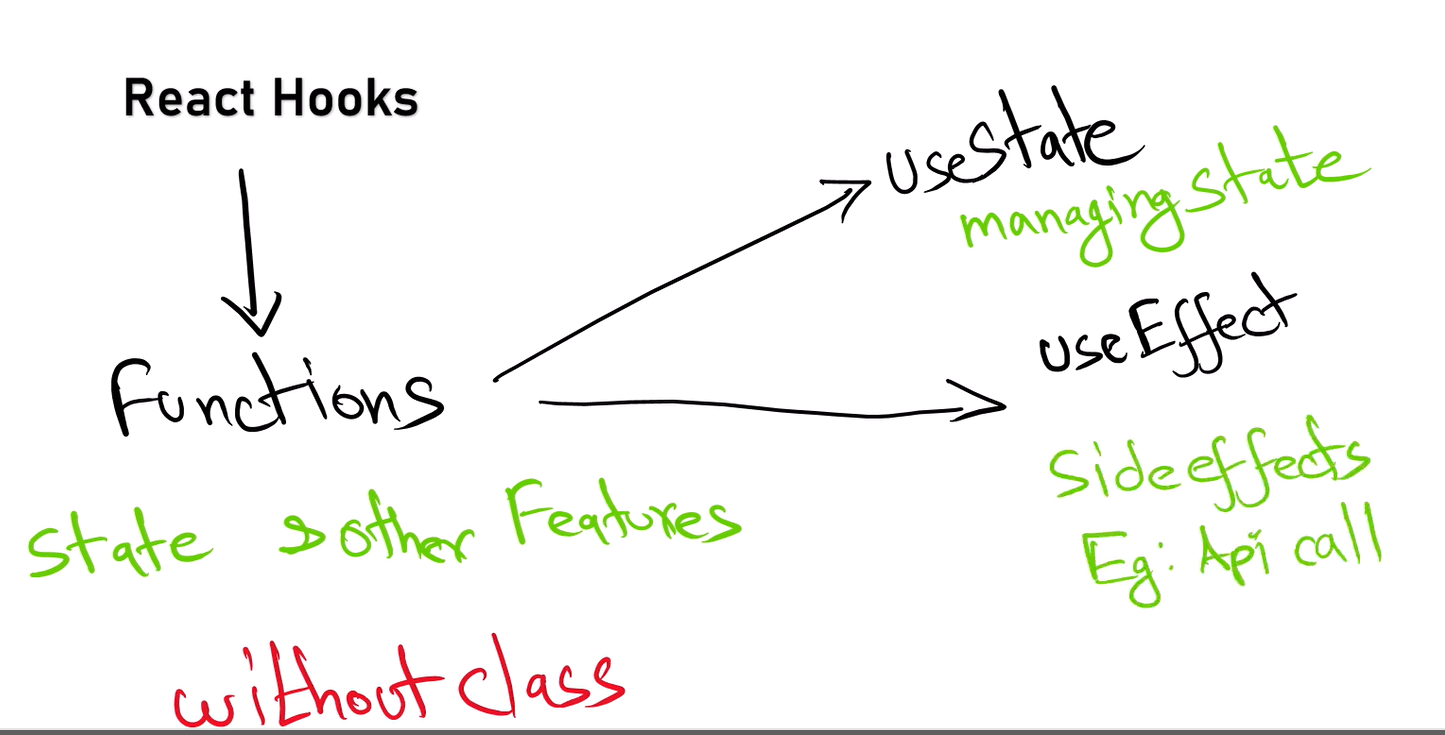
**Arrow function expression in React.js**

Arrow function is a way of declaring the function using the arrow operator where first pass the parameters in the bracket then put the whole logic.

Difference we don’t need to add function and function name.

Assigning function itself to a variable.

**React hooks**



**Lazyloading?**

Lazy loading in React helps optimize performance by loading components or resources only when they are needed. You can use React.lazy and Suspense for components, dynamic imports for assets, and libraries like react-lazy-load-image-component for images. This technique reduces initial load time and improves user experience.

**Parent to child pros**

Passing data from parent to child components in React provides benefits such as centralized data management, controlled data flow, component reusability, simplified state management, enhanced readability, encapsulation, and flexibility. This approach aligns with React’s design principles and helps build maintainable and scalable applications

**what is the disadvantages of using JavaScript in reactjs?**

If you developed react js using Javascript and we declare an object variable with string data type this will throw error at run time in production browser so will face type safety with javascript.

**Authentication flow in React.js**

Authentication Context: Use the React Context API to manage authentication state and provide it throughout your application.

Authentication Components: Create login and logout components and handle user authentication.

Protect Routes: Use a ProtectedRoute component to restrict access to certain routes based on authentication state.

Persistence: Persist authentication state using localStorage or sessionStorage.

API Integration: Implement API calls for login and manage authentication tokens as needed.

**Side effects?**

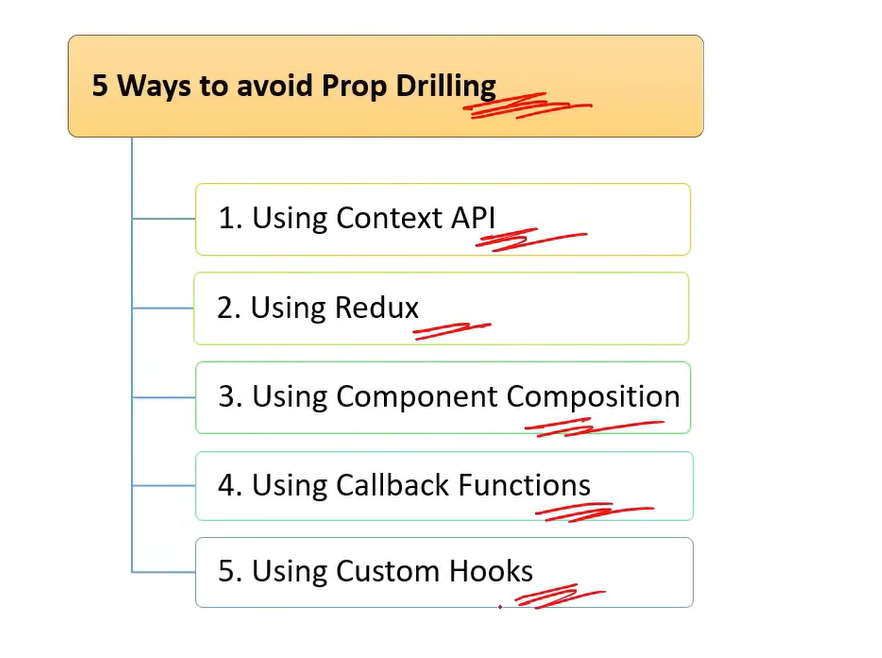
While the component is re-rendering the logic inside the component will execute again and again, in this case we have to isolate that particular logic using use Effect.

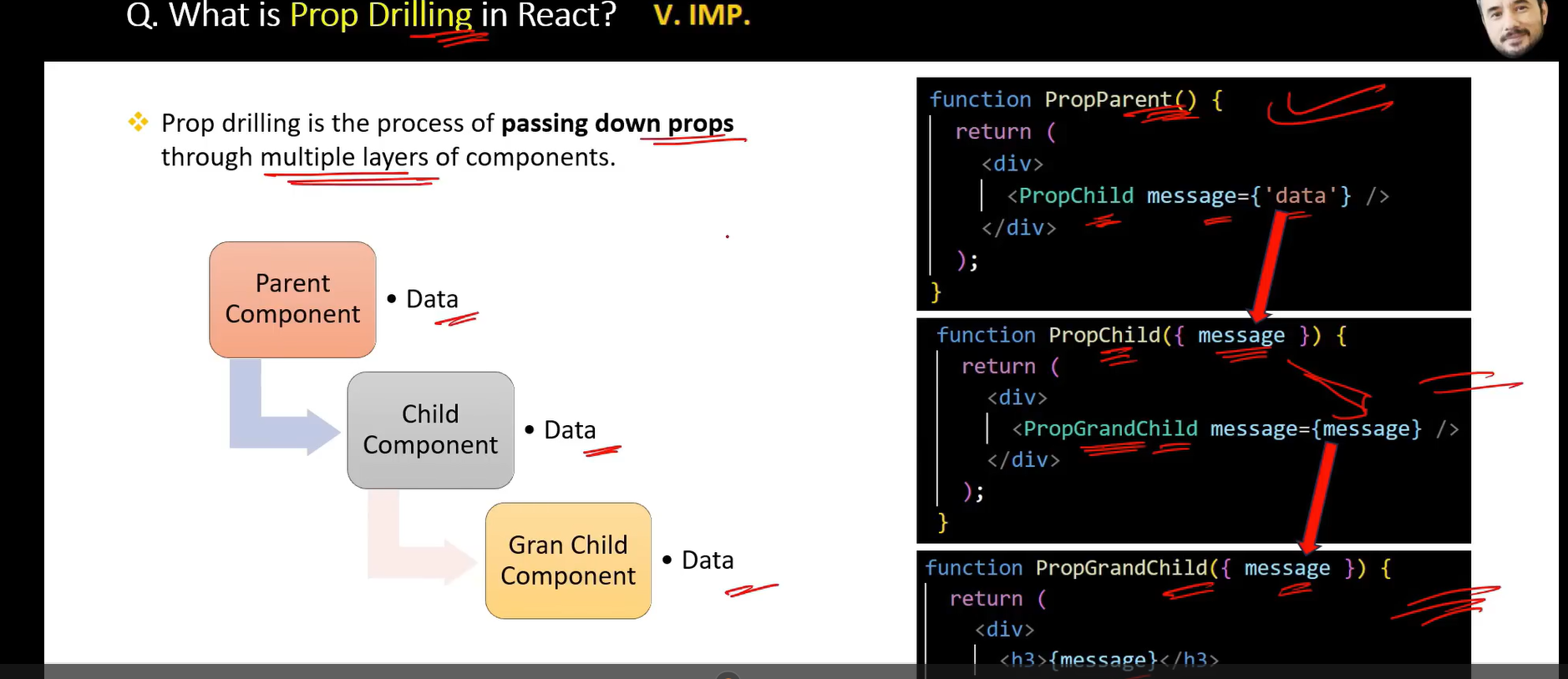
**What is Props drilling in React and how to overcome.**

Passing of props to the multi-layer of components (Nested components) or is called props drilling but the properties passing may not useful for all components that is wasting of memory

Complexity increases code reusability.

Debugging becomes challenging in in order to overcome this we have use Context. Or using redux library we can avoid props drilling.





**How to protect a Route.?**

Protected route is used to handle the security which is avail from React router dom.

**Higher order component.?**

The folder and structures maintaining in React with architecture format. The application will be scalable.

**Stateless and stateful components.?**

**Stateless Components:**

* Do not manage internal state.
* Render UI based on props.
* Simpler and more focused on presentation.

**Stateful Components:**

* Manage internal state and lifecycle methods (class components) or side effects (functional components with hooks).
* Handle user interactions and complex operations.
* More complex and capable of managing state and side effects.

**Purpose of React render Method?**

The render method is crucial in React for defining the component’s UI based on its state and props. It:

* Describes the structure of the UI.
* Updates the UI when state or props change.
* Composes complex UIs using other components.
* Operates as a pure function, ensuring predictable UI updates.

**Controlled and uncontrolled components?**

Controlled Components:

* State Management: Form data is managed via React state.
* Predictable: The component’s state is the single source of truth.
* Event Handling: Input changes update React state.
* Advantages: Easier state management, validation, and debugging.

Uncontrolled Components:

* Ref Management: Form data is managed through the DOM.
* Less Control: React does not directly manage form data.
* Event Handling: Values are accessed directly from the DOM.
* Advantages: Less boilerplate, direct access to DOM values.

**What are Pure Components?**

\*`React.PureComponent`\* is exactly the same as \*`React.Component`\* except that it handles the `shouldComponentUpdate()`, the component will re-render by default whenever `shouldComponentUpdate` is called.

a **Pure Component** is a component that only re-renders when its props or state change.

**What is lifting stateup?**

Lifting state up involves moving state management to the closest common ancestor of components that need to share the state. This pattern helps in maintaining a single source of truth, synchronizing data between components, and simplifying state management in React applications.

**"How do you handle state management in a React application? Can you describe some methods or libraries you’ve used for state management, such as Context API or Redux?"**

`useState` is indeed a fundamental hook for managing state in functional components, but there are various other methods and libraries for handling state management in React applications, especially as the application grows in complexity. Here’s a more detailed view:

1. \*\*Local State with `useState`:\*\*

- Ideal for simple state management within a single compo

nent.

- Example:

```jsx

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

```

2. \*\*Context API:\*\*

- Useful for managing state that needs to be shared across multiple components without prop drilling.

- Provides a way to pass data through the component tree without having to pass props manually at every level.

- Example:

```jsx

const MyContext = React.createContext();

function App() {

return (

<MyContext.Provider value={/\* some value \*/}>

<ChildComponent />

</MyContext.Provider>

);

}

function ChildComponent() {

const contextValue = useContext(MyContext);

return <div>{contextValue}</div>;

}

```

3. \*\*Redux:\*\*

- A state management library for more complex state management needs.

- Uses a global store to manage the state of the entire application, with actions and reducers to handle state changes.

- Useful for larger applications with complex state interactions.

- Example:

```jsx

import { createStore } from 'redux';

import { Provider, useSelector, useDispatch } from 'react-redux';

const initialState = { count: 0 };

const reducer = (state = initialState, action) => {

switch (action.type) {

case 'INCREMENT':

return { count: state.count + 1 };

default:

return state;

}

};

const store = createStore(reducer);

function App() {

return (

<Provider store={store}>

<Counter />

</Provider>

);

}

function Counter() {

const count = useSelector(state => state.count);

const dispatch = useDispatch();

return (

<div>

<p>{count}</p>

<button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>

</div>

);

}

```

**"How do you optimize the performance of a React application? Are there specific techniques or tools you use to ensure that your application runs efficiently?"**

**No problem! Performance optimization in React can be approached from several angles. Here are some key techniques and tools you can use to enhance the performance of a React application:**

1. \*\*React.memo:\*\*

- `React.memo` is a higher-order component that memoizes the result of a component rendering. It helps prevent unnecessary re-renders of functional components when the props haven't changed.

- Example:

```jsx

const MyComponent = React.memo(function MyComponent(props) {

// Component logic

});

```

2. \*\*useCallback and useMemo Hooks:\*\*

- `useCallback` is used to memoize callback functions to prevent their recreation on every render.

- `useMemo` is used to memoize expensive calculations or derived values.

- Example:

```jsx

const handleClick = useCallback(() => {

// Handle click

}, [dependencies]);

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

```

3. \*\*Code Splitting:\*\*

- Code splitting allows you to load parts of your application only when they are needed, reducing the initial load time.

- Use `React.lazy` and `Suspense` for dynamic imports.

- Example:

```jsx

const LazyComponent = React.lazy(() => import('./LazyComponent'));

function App() {

return (

<React.Suspense fallback={<div>Loading...</div>}>

<LazyComponent />

</React.Suspense>

);

}

```

4. \*\*Virtualization:\*\*

- For long lists or tables, use virtualization libraries like `react-window` or `react-virtualized` to only render visible items.

- Example with `react-window`:

```jsx

import { FixedSizeList as List } from 'react-window';

function MyList({ items }) {

return (

<List height={150} itemCount={items.length} itemSize={35} width={300}>

{({ index, style }) => <div style={style}>{items[index]}</div>}

</List>

);

}

```

5. \*\*Avoid Inline Functions in JSX:\*\*

- Defining functions directly in JSX can lead to unnecessary re-renders. Instead, define functions outside the JSX and use them.

- Example:

```jsx

// Instead of:

<button onClick={() => doSomething()}>Click me</button>

// Use:

const handleClick = () => doSomething();

<button onClick={handleClick}>Click me</button>

```

6. \*\*Efficient Rendering:\*\*

- Ensure that your component renders efficiently by avoiding unnecessary state updates, minimizing re-renders, and using appropriate keys in lists.

7. \*\*React Profiler:\*\*

- Use the React Profiler to identify performance bottlenecks in your application and get insights into why components re-render.

8. \*\*Optimize Assets:\*\*

- Optimize images, scripts, and stylesheets to reduce load times.

Would you like to explore any of these techniques in more detail or move on to the next question?

**Dynamic Imports with React.Lazy?**

* React.lazy: Allows you to load components dynamically, enabling code splitting.
* Dynamic import(): The mechanism behind React.lazy for asynchronous module loading.
* React.Suspense: Manages the loading state for components loaded lazily.
* Error Boundaries: Handle potential errors in loading components to improve user experience.

**Using React.lazy and dynamic imports effectively can help improve the performance of your React application by reducing the initial bundle size and loading components only when they are needed.**

**Sure! Let's dive into one of the techniques.**

**### Technique: Code Splitting**

**\*\*Code Splitting\*\* helps improve the performance of your React application by loading only the necessary code when it’s needed. This can reduce the initial load time of your application and improve user experience.**

**#### \*\*How to Implement Code Splitting:\*\***

**1. \*\*Dynamic Imports with React.lazy:\*\***

**- `React.lazy` allows you to load a component lazily when it’s needed.**

**- You wrap your lazy-loaded component in a `React.Suspense` component, which allows you to specify a fallback UI (e.g., a loading spinner) while the component is being loaded.**

**\*\*Example:\*\***

**```jsx**

**import React, { Suspense } from 'react';**

**const LazyComponent = React.lazy(() => import('./LazyComponent'));**

**function App() {**

**return (**

**<div>**

**<Suspense fallback={<div>Loading...</div>}>**

**<LazyComponent />**

**</Suspense>**

**</div>**

**);**

**}**

**```**

**2. \*\*Dynamic Import with React Router:\*\***

**- If you’re using React Router for navigation, you can also dynamically import route components.**

**\*\*Example:\*\***

**```jsx**

**import React, { Suspense } from 'react';**

**import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';**

**const HomePage = React.lazy(() => import('./HomePage'));**

**const AboutPage = React.lazy(() => import('./AboutPage'));**

**function App() {**

**return (**

**<Router>**

**<Suspense fallback={<div>Loading...</div>}>**

**<Switch>**

**<Route path="/home" component={HomePage} />**

**<Route path="/about" component={AboutPage} />**

**</Switch>**

**</Suspense>**

**</Router>**

**);**

**}**

**```**

**3. \*\*Webpack Code Splitting:\*\***

**- For more advanced scenarios, you can configure Webpack to split code into separate bundles based on your application's needs. This usually involves setting up Webpack’s optimization options, like `splitChunks`.**

**\*\*Example Configuration (Webpack):\*\***

**```js**

**module.exports = {**

**optimization: {**

**splitChunks: {**

**chunks: 'all',**

**},**

**},**

**};**

**```**

**#### \*\*Benefits of Code Splitting:\*\***

**- \*\*Improved Performance:\*\* Reduces the amount of JavaScript needed on the initial page load.**

**- \*\*Faster Load Times:\*\* Users only load the components they need, which can lead to quicker interactions.**

**- \*\*Enhanced User Experience:\*\* By deferring the loading of non-critical components, users see meaningful content faster.**

**FORMs?**

React Forms are the components used to collect and manage the user inputs.

**Events?**

React Events are user interactions with the web application

**Difference in ES5 ES6?**

* ES5: The version before ES6, with more basic features and less syntactic sugar.
* ES6: Introduced many new features, including let, const, arrow functions, classes, modules, template literals, destructuring, default parameters, rest and spread operators, and promises.

**Bundle of react?**

Bundling in React involves combining JavaScript, CSS, and other assets into optimized files to enhance performance and reduce the number of HTTP requests. Tools like Webpack, Vite, and Parcel are commonly used to handle this process. Bundling not only helps in optimizing load times but also in managing and organizing code effectively.