Q1: What is ReactJs

ReactJs is a javascript libray which is used for creating single page application

Q2: Advantages of ReactJs

1. React follow JSX syntax that allow developers to write HTML inside the javascript
2. It uses Virtual DOM instead of Real DOM because Real DOM manipulation is expensive
3. Uses reusable UI components to develop the UI

Q3: Virtual DOM v/s Real DOM

Virtual DOM is the light weight copy of the real DOM and it is faster than Real DOM because when there is any small changes in the UI like a small button the Real DOM repaint the whole DOM tree but what virtual DOM do is simply change the specific component in the DOM tree it does'nt repaint the whole DOM tree

Q4: Controlled v/s Uncontrolled Component

Controlled and Uncontrolled component terms are used in form elements in input.
Controlled Components are the component where forms are under the controlled of the react, means form data is handled by the state and we change that data using onChange
But in Uncontrolled component the form data is not handled by the react it directly managed by the DOM itself

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

const ControlledComponent = () => {
  const [inputValue, setInputValue] = useState('');

  const handleChange = (event) => {
    setInputValue(event.target.value);
  };

  return (
    <input
      type="text"
      value={inputValue}
      onChange={handleChange}
    />
  );
};
```

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

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

  const handleSubmit = () => {
    // Access input value using the ref
    console.log('Input value:', inputRef.current.value);
  };

  return (
    <>
      <input type="text" ref={inputRef} />
      <button onClick={handleSubmit}>Submit</button>
    </>
  );
};
```

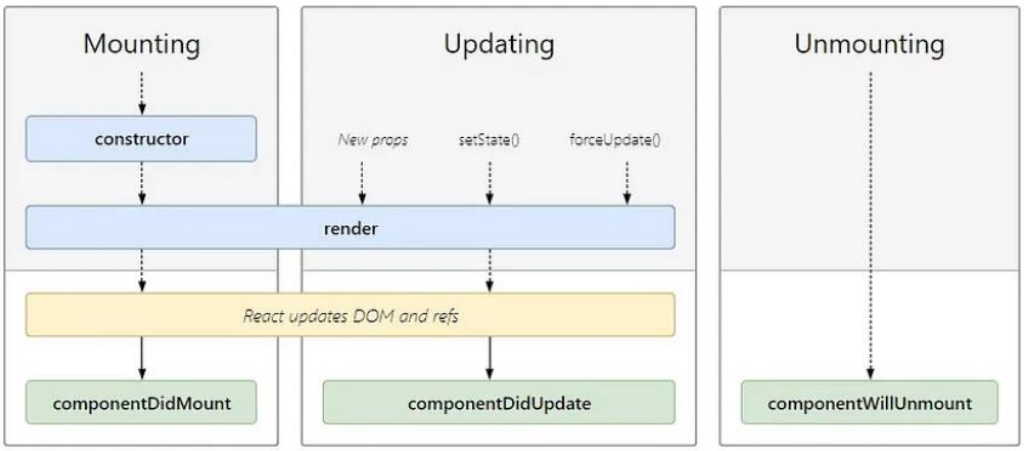
Q5: Explain React life cycle methods

There are three phase in react life cycle methods: Mounting, Updating and Unmounting

(i) inside the mounting phase there are two methods constructor method which is used for initialize the state and the another one is the componentDidMount method which runs only one time after component render this method is also used for side effects like data fetching

(ii) inside the updating phase there is a method called componentDidUpdate which runs whenever any changes in state or props

(iii) inside the unmounting phase there is a method called componentWillUnmount which runs whenever the component is removed from the DOM



Common lifecycles

Q6: Higher Order Component

Higher Order component is a component that takes another component as an argument and return a new component with additional behavior. It allows us to share the common logic or functionality among the multiple component

App.js

```
import ClickCounter from "./ClickCounter";
import HoverCounter from "./HoverCounter";
import "./styles.css";

export default function App() {
  return (
    <div className="App">
      <ClickCounter />
      <HoverCounter />
    </div>
  );
}
```

HocCounter.js

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

const HocCounter = (OriginalComponent) => {
  const EnhancedComponent = () => {
    const [count, setCount] = useState(0);
    const incrementCount = () => {
      setCount(count + 1);
    };
    return <OriginalComponent count={count} incrementCount={incrementCount} />;
  };
  return EnhancedComponent;
};

export default HocCounter;
```

ClickCounter.js

```
import HocCounter from "./HocCounter";

const ClickCounter = ({ count, incrementCount }) => {
  return <button onClick={incrementCount}>Click {count}</button>;
};

export default HocCounter(ClickCounter);
```

HoverCounter.js

```
import HocCounter from "./HocCounter";

const HoverCounter = ({ count, incrementCount }) => {
  return <button onMouseOver={incrementCount}>Hover {count}</button>;
};

export default HocCounter(HoverCounter);
```

Q7: Api calling in ReactJs

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

const App = () => {
  const [userTitle, setUserTitle] = useState([]);

  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/posts")
      .then((res) => {
        setUserTitle(res.data);
      })
      .catch((err) => console.log(err));
  }, []);

  return (
    <>
      {userTitle?.map((item, id) => (
        <p key={id}>{item.title}</p>
      ))}
    </>
  );
};

export default App;
```

Q8: Pure Component

Pure component is used for performance optimization means it improve the re-rendering of the component … for example if there is any changes in the props or the state than only the component will render else it will not render

You can achieve the pure component in class based component using React.PureComponent instead of React.Component

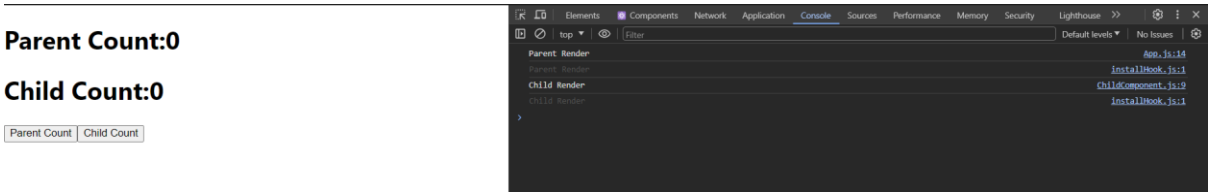
App.js

```jsx
import React from "react";
import "./App.css";
import ChildComponent from "./ChildComponent";

You, 13 minutes ago | 1 author (You)
class App extends React.PureComponent {
  constructor() {
    super();
    this.state = {
      count: 0,
      count2: 0,
    };      You, 13 minutes ago • Uncommitted changes
  }
  render() {
    console.log("Parent Render");
    return (
      <>
        <h1>Parent Count:{this.state.count}</h1>
        <ChildComponent count={this.state.count2} />
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Parent Count
        </button>
        <button
          onClick={() => this.setState({ count2: this.state.count2 + 1 })}
        >
          Child Count
        </button>
      </>
    );
  }
}
```

ChildComponent.js

```jsx
import React from "react";

class ChildComponent extends React.PureComponent {
  constructor() {
    super();
  }

  render() {
    console.log("Child Render");
    return <h1>Child Count:{this.props.count}</h1>;
  }
}

export default ChildComponent;
```

Output Screen



Here when user click on the Parent Count button than only Parent Render will console on the screen because there is no change in the child component but when user click on the Child Count than Parent Render and Child Render both will console and if you don't write React.PureComponent in ChildComponent than if you click on the Parent Count button than Parent Render and Child Render both will console

Q9: Pure component in Functional Component

You can achieve pure component in functional component using React memo

App.js

```jsx
import React, { useState } from "react";
import ChildComponent from "./ChildComponent";

const App = () => {
  const [count, setCount] = useState(0);
  const [count2, setCount2] = useState(0);

  console.log("Parent Render");
  return (
    <>
      <h1>Parent Count:{count}</h1>
      <ChildComponent count={count2} />
      <button onClick={() => setCount(count + 1)}>Parent Count</button>
      <button onClick={() => setCount2(count2 + 1)}>Child Count</button>
    </>
  );
};

export default App;
```

ChildComponent.js

```jsx
import React, { memo } from "react";

const ChildComponent = ({ count }) => {
  console.log("Child Render");
  return <h1>Child Count:{count}</h1>;
};

export default memo(ChildComponent);
```

Q10: Explain useContext

useContext is used to share data between components without passing props in each level ……
there are three steps to create useContext

1. Create the context
2. Provide the context with the value using Provider
3. Consume the context value

App.js

```jsx
import React, { createContext } from "react";
import ComponentA from "./ComponentA";

export const StoreContext = createContext();

const userData = {
  name: "Samir Singh",
  designation: "Software Engineer",
};

const App = () => {
  return (
    <>
      <StoreContext.Provider value={userData}>
        <ComponentA />
      </StoreContext.Provider>
    </>
  );
};

export default App;
```

ComponentA.js

```jsx
import React from "react";
import ComponentB from "./ComponentB";

const ComponentA = () => {
  return (
    <>
      <ComponentB />
    </>
  );
};

export default ComponentA;
```

ComponentB.js

```jsx
import React from "react";
import ComponentC from "./ComponentC";

const ComponentB = () => {
  return (
    <>
      <ComponentC />
    </>
  );
};

export default ComponentB;
```

ComponentC.js

```jsx
import React, { useContext } from "react";
import { StoreContext } from "./App";

const ComponentC = () => {
  const userData = useContext(StoreContext);

  return <h1>{`${userData?.name} - ${userData?.designation}`}</h1>;
};

export default ComponentC;
```

Here inside the App.js we create the context and pass the value using provider and inside the ComponentC.js we consume the context value

Q11: Functional component v/s Class component

Functional component:

1. Functional components are written as a JavaScript function.
2. Functional components do not have any life cycle methods.
3. Functional components are faster than class component because it need to do less work than class component like initializing constructor and render method
4. The syntax of functional component is easy than class component.

Class component:

1. Class components are written as a JavaScript Class.
2. Class components have life cycle methods like componentDidMount, componeneDidUpdate etc.
3. Class components are slower than functional component because it need to do more work than functional component.
4. The syntax of class component is complicated as compare to functional component.

Q12: Explain Redux Flow

Redux is a state management javascript library which is used to handle state globally through out the component. There are some concepts inside the redux

1. Store: Redux uses a single store to hold the entire state of the application. The store is basically an object that holds the state.
2. Actions: Actions are plain javascript object which is called on an event
3. Reducers: Reducers are functions that takes two arguments first one is the current state and second one is the action and returns the new state
4. Dispatch: Dispatch is a method which is used to call the actions

Now in our code when user click on any button the action will dispatch, this action will sent to the reducer and than reducer change return the new state and redux update the state in the store with the new state and than that state update in our UI. ***This flow creates the bidirectional flow in reactjs***

Q13: Explain useMemo Hook

useMemo Hook is used for memoize the value. It is used for performance optimization in which the expensive calculation runs only when the dependency of useMemo change

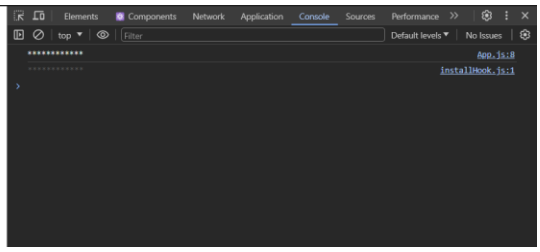
```jsx
import React, { useMemo, useState } from "react";

const App = () => {
  const [add, setAdd] = useState(0);
  const [sub, setSub] = useState(100);

  const multiply = useMemo(() => {
    console.log("************");
    return add * 5;
  }, [add]);
      You, 3 weeks ago • Initialize project using Create React App
  return (
    <>
      <h1>Multiply : {multiply}</h1>
      <h1>Addition : {add}</h1>
      <h1>Subtraction : {sub}</h1>

      <button onClick={() => setAdd(add + 1)}>Addition</button>
      <button onClick={() => setSub(sub - 1)}>Subtraction</button>
    </>
  );
};

export default App;
```

Multiply : 0

Addition : 0

Subtraction : 100

Addition | Subtraction



Here when user click on the addition button then only multiply function will run and when user click on the subtraction button than multiply function will not run

Q14: SSR v/s CSR

SSR:

1. SSR is the another way of displaying web pages on the browser. In SSR the server generates HTML for web pages and sends the fully rendered page to the client to display the page

2. SSR is better for SEO (Search Engine Optimization) because web crawlers crawls the server side website easier

3. It is great for static websites not for dynamic like animations and interactive media

4. Some examples of server side frameworks are Next.js, Nest.js, Nuxt.js

CSR:

1. CSR is the another way of displaying web pages on the browser. In CSR the web browser takes the responsibility of generating the HTML web pages

2. CSR is not great for SEO

3. It is great for dynamic and interactive sites

4. Some examples of client side frameworks are React.js, Angular.js, Vue.js

Q15: Explain Lazy Loading

Lazy Loading is a technique in reactjs which is used for improve the performance of web appplications.
Lazy Loading is used to load only those component which shows on viewport
This can help reduce the initial bundle size and improve the overall load time of the application, especially for larger applications with many components
*It follows the **code splitting** technique which is used for breaking the code into smaller chunks and load the each chunk separately when needed*

```jsx
import React from "react";

const LazyAbout = React.lazy(() => import("./About"));

const App = () => {
  return (
    <>
      <React.Suspense fallback="Loading...">
        <LazyAbout />
      </React.Suspense>
    </>
  );
};

export default App;
```

Q16: Some techniques to optimize performance of the react application

1. Memoization with React.memo() : Use React.memo() higher order component to memoize the functional components.
2. useMemo and useCallback : Use this hooks to memorize the values and function, to prevent unnecessary re-renders.
3. Code splitting : Use React.lazy and suspense to improve the initial loading time of the application

Q17: Explain useCallback Hook

useCallback hook is used for memoize the function means suppose when you pass the function from parent to child and if there is any change in parent component than the child component still renders because of passing function to the child component so for that you have to use useCallback hook, because the memoize only checks the shallow copy means if the value is same but the reference is different than still the component will re-render because the function always create new reference on new render of the component

Below you can see the example of useCallback here remove the useCallback and check even after using React.memo when you change the input value the ChildCounter component will render because when the state change from parent component it creates the new reference of the updateCounter function that's why we have to wrap this function into useCallback

App.js

```jsx
import React, { useCallback, useState } from "react";
import ChildCounter from "./ChildCounter";

const App = () => {
  const [count, setCount] = useState(0);
  const [inputValue, setInputValue] = useState("");

  const updateCounter = useCallback(() => setCount(count + 1), [count]);

  return (
    <>
      <div>
        <h1>Parent Count - {count}</h1>
        <button onClick={() => setCount(count + 1)}>Parent Counter</button>
        <input
          type="text"
          value={inputValue}
          onChange={(e) => setInputValue(e?.target?.value)}
        />

        <ChildCounter count={count} updateCounter={updateCounter} />
      </div>
    </>
  );
};

export default App;
```

ChildCounter.js

```jsx
import React, { memo } from "react";

const ChildCounter = ({ count, updateCounter }) => {
  console.log("child render");
  return (
    <div>
      <h1>Child Counter - {count}</h1>
      <button onClick={updateCounter}>Child Counter</button>
    </div>
  );
};

export default memo(ChildCounter);
```

Output Screen