REACT INTERVIEW QUESTIONS (DETAILED ANSWERS)

1. What is React, and why is it popular?

Answer:
React is a JavaScript library developed by Facebook for building user interfaces, primarily for single-page applications (SPAs). It enables developers to create reusable UI components and efficiently update and render UI elements based on data changes.

Reasons for its popularity:

- **Virtual DOM**: React uses a Virtual DOM to optimize rendering, making updates more efficient.
- **Component-Based Architecture**: Code is broken into reusable components, making it easier to manage.
- **Unidirectional Data Flow**: Helps maintain a predictable state.
- **Performance Optimization**: Uses a diffing algorithm to update only necessary parts of the UI.
- **Strong Community Support**: Backed by Facebook and widely adopted by developers.
- **Flexibility**: Works with other libraries like Redux for state management.

---

2. Advantages and Disadvantages of React

Advantages:

- **Fast and Efficient Rendering**: Uses Virtual DOM to update only changed components.
- **Reusable Components**: Enhances modularity and maintainability.
- **Strong Community Support**: Plenty of open-source libraries and third-party tools.
- **Easy to Learn**: If you know JavaScript, learning React is straightforward.
- **SEO Friendly**: Supports server-side rendering (SSR), improving SEO performance.
- **Flexibility**: Can be integrated with other libraries and frameworks like Next.js.

Disadvantages:

- **JSX Learning Curve**: JSX (JavaScript XML) syntax can be tricky for beginners.
- **Fast-Paced Updates**: Frequent updates require developers to stay updated.
- **State Management Complexity**: Large applications may require state management tools like Redux or Context API.
- **Poor Documentation at Times**: Some features lack detailed documentation.

---

3. Explain the concept of Virtual DOM & Real DOM

Answer:

- **Real DOM (Document Object Model)** represents the actual UI elements in the browser. Every time a change occurs, the entire DOM is updated, making it slow and inefficient.
- **Virtual DOM** is a lightweight copy of the Real DOM maintained by React. When changes occur, React updates the Virtual DOM first, compares it with the previous version (diffing), and then updates only the changed parts in the Real DOM. This improves performance and optimizes rendering.

---

4. What are components in React?

Answer:
Components are independent, reusable pieces of UI in React. They can be classified into:

- **Functional Components** (Stateless) – Defined as functions and can use hooks for state management.
- **Class Components** (Stateful) – Defined as ES6 classes and can manage their own state.

Example of a functional component:

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

Example of a class component:

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

---

5. What is the difference between functional and class components?

Feature	**Functional Component**	**Class Component**
Syntax	JavaScript function	ES6 class
State Management	Uses Hooks (useState, useEffect)	Uses this.state
Performance	Faster due to no lifecycle methods	Slightly slower due to lifecycle methods
Lifecycle Methods	No built-in lifecycle methods	Has lifecycle methods like componentDidMount
Simplicity	Shorter, cleaner code	More complex due to this keyword

---

6. What are props in React?

Answer:
Props (short for properties) are used to pass data from a parent component to a child component. They are read-only and cannot be modified by the child component.

Example:

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

// Usage
<Welcome name="Komal" />
```

---

7. How does state work in React?

Answer:
State is a built-in React object that holds dynamic data about a component. Unlike props, state is mutable and can change over time.

Example using useState hook in a functional component:

```
import { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

In class components, state is managed using this.state and this.setState().

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

---

8. What is the difference between state and variables?

Answer:

Feature	**State**	**Variable**
Scope	Belongs to React component	Local to the function
Persistence	Retains value between renders	Does not persist after function execution
Updates	Causes component to re-render	Does not trigger re-render
Management	Managed using useState or this.state	Managed normally in JavaScript

Example:

```
// Using state (Triggers re-render)
const [count, setCount] = useState(0);
setCount(count + 1);

// Using variable (No re-render)
let count = 0;
count += 1;
```

---

9. What is the purpose of render() in React?

Answer:

- In class components, render() is a lifecycle method responsible for rendering UI elements.
- It returns the JSX to be displayed in the UI.
- It is called automatically whenever state or props change.

Example:

jsx

CopyEdit

```
class App extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
```

---

10. Explain lifecycle methods in React.

Answer:
Lifecycle methods are special functions in class components that execute at different phases of a component’s life.

Phases:

1. **Mounting (Component Creation)**
   - constructor()
   - static getDerivedStateFromProps()
   - render()
   - componentDidMount()
2. **Updating (Re-rendering due to state/props change)**
   - static getDerivedStateFromProps()
   - shouldComponentUpdate()
   - render()
   - getSnapshotBeforeUpdate()
   - componentDidUpdate()
3. **Unmounting (Component Removal)**
   - componentWillUnmount()

Example:

```
class LifecycleDemo extends React.Component {
  componentDidMount() {
    console.log("Component Mounted!");
  }


  componentWillUnmount() {
    console.log("Component Unmounted!");
  }
  render() {
    return <h1>Lifecycle Example</h1>;
  }
}
```

---

11. What are hooks in React?

Answer:
Hooks allow functional components to use state and lifecycle features without converting to class components.

Common hooks:

- useState (Manages state)
- useEffect (Performs side effects)
- useContext (Uses Context API)
- useRef (Accesses DOM elements)
- useMemo (Optimizes performance)

---

12. Explain useState and useEffect hooks.

- **useState**:
  - useState is used to declare state variables in functional components.
  - It returns an array with two values: the current state and a function to update it.

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

setCount(count + 1);
```

- **useEffect**:
  - useEffect is used for side effects like fetching data, subscriptions, or manually updating the DOM in functional components.
  - It runs after the render and can be triggered by specific dependencies.

```
useEffect(() => {

  console.log("Component mounted or updated!");

}, [count]);  // Dependency array: Effect runs when 'count' changes.
```

---

13. How do you handle forms in React?

Answer:
Forms in React are handled by using controlled components, where the form data is managed by the component's state. The value of the input field is tied to the state, and the state is updated with every change.

- **Example of a controlled form**:

```
function MyForm() {
  const [name, setName] = useState('');

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

  const handleSubmit = (event) => {
    event.preventDefault();
    alert('Name submitted: ' + name);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={name} onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

---

14. What are controlled and uncontrolled components?

- **Controlled Components**:
  - The form elements' values are controlled by the React component's state.
  - Example: <input value={state} onChange={updateState} />
- **Uncontrolled Components**:
  - The form elements manage their own state internally. They don't require useState.
  - Typically use the ref attribute to get values.
  -

```
const inputRef = useRef();


const handleSubmit = () => {
  alert(inputRef.current.value);
};


return <input ref={inputRef} />;
```

---

15. What are Dumb Components, and Sibling Components?

- **Dumb Components**:
  - Also known as **Presentational Components**.
  - They are concerned only with displaying data passed via props and don’t manage state or logic.

```
function Display(props) {
  return <h1>{props.message}</h1>;
}
```

- **Sibling Components**:
  - Components that are at the same level in the component hierarchy.
  - These components can share data via their common parent.

```
function ParentComponent() {
  const [sharedData, setSharedData] = useState('Data');
  return (
    <div>
      <Sibling1 data={sharedData} />
      <Sibling2 setData={setSharedData} />
    </div>
  );
}
```

---

16. How do you optimize performance in a React application?

Answer:

- **Memoization**: Use React.memo() to prevent re-rendering components that haven't changed.
- **useMemo**: Memorize expensive calculations and return the cached value to avoid recalculating.
- **useCallback**: Memorize functions to prevent unnecessary re-creation during re-renders.
- **Lazy loading**: Dynamically load components using React.lazy().
- **Code splitting**: Split large files into smaller chunks using tools like Webpack.
- **Virtualization**: For large lists, use libraries like react-window to render only the visible items.

---

17. What is React Router?

Answer:
React Router is a library that enables routing in React applications. It allows navigating between different components and views without reloading the page (SPA behavior). React Router provides components like <Route>, <Link>, and <BrowserRouter> to manage client-side routing.

---

18. How do you implement routing in a React app?

Answer:
First, install react-router-dom:

Bash-

```
npm install react-router-dom
```

Then, use it in your app:

jsx

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


function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Switch>
      <Route path="/" exact component={Home} />

      <Route path="/about" component={About} />

     </Switch>

   </Router>

  );

}
```

---

19. What are Route and Link components?

- **Route**:
    - Defines which component should be rendered based on the URL path. It's used to associate a path with a React component.

```
<Route path="/about" component={About} />
```

- **Link**:
    - Used for navigation in React Router. It replaces <a> tags and prevents page reloads.

```
<Link to="/about">Go to About Page</Link>
```

---

20. What is the difference between useNavigate and Link?

Answer:

- **useNavigate** is a hook that allows imperative navigation, meaning you can programmatically change routes (without a <Link>).

```
const navigate = useNavigate();

navigate('/about');
```

- **Link** is a declarative way to navigate by clicking on a link.

```
<Link to="/about">Go to About</Link>
```

---

21. What is Redux, and how does it work with React?

Answer:
Redux is a state management library that helps manage the state of the application globally. It is commonly used with React to centralize and share state across components.

- **How it works**:
  Redux uses a **store** that holds the entire state of the application. Actions are dispatched to trigger state changes, and reducers handle those actions to update the store. React components connect to Redux via useSelector and useDispatch to read from the store and dispatch actions.

---

22. What are actions, reducers, payload, useSelector, useDispatch, and State in Redux?

- **Actions**:
    - Plain JavaScript objects that describe an event or intention to update the state.

```
const action = { type: 'INCREMENT' };
```

- **Reducers**:
    - Pure functions that specify how the state changes based on the action dispatched.

```
function counterReducer(state = 0, action) {
 switch (action.type) {
  case 'INCREMENT':
   return state + 1;
  default:
   return state;
 }
}
```

- **Payload**:
    - Additional data passed along with an action. It's often used to update the state.

```
const action = { type: 'SET_NAME', payload: 'John' };
```

- **useSelector**:
    - A hook used to access the state from the Redux store.

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

- **useDispatch**:
    - A hook used to dispatch actions to the Redux store.

```
const dispatch = useDispatch();
dispatch({ type: 'INCREMENT' });
```

- **State**:
    - The central store that holds the application's global state.

---

23. What is memoization, and how is it used in React?

Answer:
Memoization is an optimization technique used to speed up repeated function calls by caching the result of expensive operations. In React, this can be achieved using React.memo(), useMemo(), and useCallback().

- **React.memo()**:
    - A higher-order component that prevents unnecessary re-renders of functional components by shallowly comparing props.

```
const MemoizedComponent = React.memo(MyComponent);
```

- **useMemo()**:
    - Used to memoize expensive calculations, returning a cached value unless dependencies change.

```
const expensiveValue = useMemo(() => computeExpensiveValue(data), [data]);
```

- **useCallback()**:
    - Memorizes a function to prevent its re-creation on every render.

```
const memoizedCallback = useCallback(() => { console.log('Hello'); }, []);
```

---

24. Explain the concept of React's PureComponent.

Answer:
PureComponent is a base class for components that automatically implement a **shallow comparison** of props and state. If there's no change, it prevents unnecessary re-renders. This can improve performance when a component's props and state do not change frequently.

- **Shallow Comparison**:
  React compares only the values of props and state, not nested objects, to decide whether to re-render.

```
class MyComponent extends React.PureComponent {
  render() {
    return <div>{this.props.name}</div>;
  }
}
```

---

25. How do you prevent unnecessary re-renders in React?

Answer:
There are several ways to prevent unnecessary re-renders in React:

- **React.memo()**: Wrap functional components to prevent re-rendering when props haven't changed.
- **PureComponent**: Use PureComponent to ensure that the component only re-renders when props or state change.
- **shouldComponentUpdate**: Implement this lifecycle method in class components to manually control when the component should re-render.
- **Memoization with useMemo and useCallback**: Memoize calculations or functions to avoid recalculating or recreating them on every render.
- **Optimize state updates**: Avoid unnecessary state changes. Use setState or useState only when necessary.

---

26. What testing frameworks do you use with React (What is Jest)?

Answer:

- **Jest** is a popular testing framework often used in conjunction with React. It's built by Facebook and provides a simple, zero-config test runner with built-in assertion functions. It also supports mocking and snapshot testing.

Key features of Jest:

- Supports **unit tests**, **integration tests**, and **snapshot tests**.
- Built-in test runner with support for parallel test execution.
- **Mocking functions** to simulate the behavior of components or APIs.
- **Snapshot testing** to verify that components' outputs remain consistent over time.

- **React Testing Library**:
  - This library focuses on testing components the way users interact with them. It encourages writing tests that resemble the real user experience.

---

27. How do you handle asynchronous operations in React?

Answer:
Asynchronous operations in React are typically handled using **async/await** in combination with the useEffect hook or lifecycle methods. Here's how you can fetch data asynchronously:

1. **Using useEffect for async operations**:

```
useEffect(() => {
  const fetchData = async () => {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    setData(data);
  };

  fetchData();
}, []);  // Empty array ensures it runs once after initial render.
```

2. **Handling Promises**:
   - You can also use .then() and .catch() if you prefer handling promises directly.

---

28. What is code splitting in React?

Answer:
Code splitting is the practice of breaking down the JavaScript code into smaller, more manageable bundles that can be loaded on demand, instead of loading everything upfront. This can drastically improve the performance of your React application.

- **React.lazy()**:
    - It is used to dynamically import components, splitting them into separate chunks that can be loaded only when needed.

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

- **Suspense**:
    - A React component that is used in combination with React.lazy() to show fallback UI while the lazy-loaded component is being fetched.

```
<Suspense fallback={<div>Loading...</div>}>
  <LazyComponent />
</Suspense>
```

---

29. Explain the concept of Higher-Order Components (HOCs).

Answer:
A **Higher-Order Component** (HOC) is a function that takes a component and returns a new component with enhanced behavior. It's a pattern used to share common functionality between components.

- HOCs don't modify the original component but return a new one.

```
function withLoading(Component) {
  return function WithLoading(props) {
    if (props.isLoading) {
      return <div>Loading...</div>;
    }
    return <Component {...props} />;
  };
}

const EnhancedComponent = withLoading(MyComponent);
```

---

30. What is useMemo and useCallback hooks?

- **useMemo**:
  - It is used to memoize expensive computations, so the calculation is only done when the dependencies change.
  - Helps avoid recalculating values on every render.

```
const memoizedValue = useMemo(() => computeExpensiveValue(input), [input]);
```

- **useCallback**:
  - It returns a memoized version of a function that only changes if one of the dependencies has changed.
  - Useful for preventing unnecessary re-creations of functions.

```
const memoizedCallback = useCallback(() => { doSomething(); }, [dependencies]);
```

---

31. Explain how you would create a custom hook.

Answer:
To create a custom hook in React:

1. Write a function that uses the built-in React hooks (like useState, useEffect, etc.).
2. Prefix the function with use to follow React's convention for hooks.
3. Return any necessary values or functions from the hook.

Example of a custom hook to track window size:

```
function useWindowSize() {
  const [size, setSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight,
  });

  useEffect(() => {
    const handleResize = () => {
      setSize({
        width: window.innerWidth,
        height: window.innerHeight,
      });
    };

    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);

  return size;
}
```

---

32. What is lazy loading, and how can it be implemented in a React application?

Answer:
Lazy loading is the technique of loading components only when they are required, instead of loading all components upfront. It can be implemented using React.lazy() for components and Suspense to handle loading states.

```
const LazyComponent = React.lazy(() => import('./LazyComponent'));
<Suspense fallback={<div>Loading...</div>}>
  <LazyComponent />
</Suspense>
```

---

33. How do you handle events in React?

Answer:
React handles events in a similar way to DOM events but uses a synthetic event system. Event handlers are passed as props to React elements and can be used to perform actions.

```
function handleClick() {
  alert('Button clicked!');
}
<button onClick={handleClick}>Click me</button>
```

- React normalizes events to ensure they work consistently across browsers.

---

34. What is event bubbling?

Answer:
Event bubbling is a concept where events propagate up from the target element to the root of the DOM. It allows parent elements to listen to events on their children by attaching a single listener.

```
<div onClick={() => alert("Div clicked!")}>
  <button onClick={(e) => e.stopPropagation()}>Click me</button>
</div>
```

---

35. What is server-side rendering (SSR)?

Answer:
Server-side rendering (SSR) refers to the process of rendering React components on the server, sending the fully rendered HTML to the client, and then hydrating it on the client-side. This can improve SEO and load time.

- Example:
  SSR can be achieved with frameworks like **Next.js**, where React components are rendered on the server and sent to the browser.

---

36. What is TypeScript, and how can it be integrated with React?

Answer:
TypeScript is a superset of JavaScript that adds static types, making it easier to catch errors at compile-time. It can be integrated with React by installing the TypeScript package and using .tsx files for components.

- Example:

Bash-

```
npm install typescript @types/react @types/react-dom
```

---

37. What are some popular UI libraries for React?

Answer:
Some popular UI libraries that integrate well with React are:

1. **Material-UI (MUI)**:
   - A popular React UI framework based on Google’s Material Design.
   - Offers pre-built, customizable components like buttons, grids, dialogs, etc.

2. **Ant Design**:
    - A comprehensive design system with a set of high-quality React components.
    - Provides ready-to-use components with an emphasis on design consistency.
3. **Chakra UI**:
    - A simple, modular, and accessible component library for React.
    - It offers a collection of accessible and customizable UI components.
4. **React Bootstrap**:
    - A React-based implementation of the Bootstrap framework.
    - Provides Bootstrap-styled components as React components.
5. **Semantic UI React**:
    - A React integration for Semantic UI, which is a CSS framework.
    - Provides a set of React components that follow the Semantic UI design principles.
6. **BlueprintJS**:
    - A React-based UI toolkit for building complex, data-dense web interfaces.
    - Great for enterprise-level applications with forms, tables, and modals.

---

38. What are the best practices for deploying a React application?

Answer:
Here are some best practices for deploying a React application:

1. **Use a build process**:
    - Ensure the application is optimized using npm run build to minimize bundle size.
    - This process generates static files that can be served from any static file server.
2. **Optimize assets**:
    - Compress images, use SVGs, and optimize CSS and JS files.
    - Use modern formats like WebP for images to reduce size without compromising quality.
3. **Use environment variables**:
    - Keep sensitive information, like API keys, in .env files and avoid hardcoding them in the codebase.
4. **Choose an appropriate hosting platform**:
    - Platforms like **Vercel**, **Netlify**, **GitHub Pages**, and **AWS Amplify** provide free or affordable hosting for React applications.

5. **Enable HTTPS**:
    - Serve your application over HTTPS to secure the data transfer between the client and server.
6. **Implement a CDN**:
    - Use a Content Delivery Network (CDN) to serve your assets to reduce latency and increase performance.
7. **SEO Optimization**:
    - Use server-side rendering (SSR) or static site generation (SSG) for better SEO, especially with tools like **Next.js**.
8. **Error Handling & Logging**:
    - Use services like **Sentry** or **LogRocket** for real-time error logging in production.

---

39. Explain JSX.

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript used with React. It allows you to write HTML-like code in your JavaScript files. This code is then transformed into JavaScript code using a tool like Babel.

- JSX enables writing components more easily and intuitively.

```
const element = <h1>Hello, world!</h1>;
```

- **Why use JSX?**
    - Makes it easier to create React components.
    - Provides a clean syntax that combines markup and logic in the same file.
    - JSX can contain expressions, and the code is compiled into React.createElement() calls.

---

40. How does React handle state management?

Answer:
React has a built-in state management system for components:

1. **State in Functional Components**:
    - Managed using the useState hook.

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

2. **State in Class Components**:
    - Managed using this.state and this.setState().

3. **State Lifting**:
    - In React, state can be "lifted" from a child to a parent component to allow for shared state.
4. **State Management Libraries**:
    - **Redux** and **Context API** are often used for more complex state management.
        - **Redux**: Centralizes the state and allows global state management across components.
        - **Context API**: Useful for passing state down the component tree without prop drilling.

---

41. What is the difference between state and props in React?

Answer:

- **State**:
    - A local data source within a component.
    - Managed and changed within the component using useState (for functional components) or this.setState() (for class components).
    - It is mutable and used to track data that affects rendering.
- **Props**:
    - Short for "properties", passed from a parent component to a child component.
    - Immutable, meaning they cannot be changed by the receiving component.
    - Used to pass data and event handlers between components.

---

42. What is the difference between setState and forceUpdate?

Answer:

- **setState()**:
    - Used to update the state of a React component and trigger a re-render.
    - It ensures that the component re-renders only when the state has changed.
- **forceUpdate()**:
    - Used to force a re-render of the component, even if the state or props haven’t changed.
    - This is generally avoided, as it bypasses React’s optimizations and can lead to performance issues.

---

43. What are the different lifecycle methods in React?

Answer:
The lifecycle methods in React are used in class components to manage different stages of a component's life. These include:

1. **Mounting**:
   - **constructor()**: Initializes the component state.
   - **componentDidMount()**: Invoked after the component is added to the DOM.
2. **Updating**:
   - **shouldComponentUpdate()**: Determines whether the component should re-render.
   - **componentDidUpdate()**: Invoked after the component has re-rendered.
3. **Unmounting**:
   - **componentWillUnmount()**: Cleanup logic before the component is removed from the DOM.
4. **Error Handling**:
   - **componentDidCatch()**: Handles errors in the component tree.

---

44. When is the componentDidMount method called?

Answer:
componentDidMount() is called **once**, immediately after the component has been rendered to the DOM. It's commonly used to perform tasks like:

- Fetching data from an API.
- Setting up subscriptions or event listeners.
- Initializing third-party libraries.

---

45. What is the purpose of the componentWillUnmount method?

Answer:
componentWillUnmount() is called **right before** the component is removed from the DOM. It is used for cleanup tasks like:

- Clearing timers or intervals.
- Removing event listeners.
- Canceling network requests.

---

46. Explain the shouldComponentUpdate method.

Answer:
shouldComponentUpdate(nextProps, nextState) is used to optimize performance. It determines whether the component should re-render based on changes in props or state.

- If it returns false, React will **skip** the re-render.
- If it returns true, React will continue with the re-render process.

---

47. What is the purpose of the BrowserRouter component?

Answer:
BrowserRouter is a component from the **React Router** library that uses the HTML5 History API to keep your UI in sync with the URL. It provides the context for routing and enables navigation between different views within a React application.

```
<BrowserRouter>
  <App />
</BrowserRouter>
```

---

48. How do you fetch data from an API in React?

Answer:
In React, you can fetch data from an API using the fetch API or libraries like **Axios**.

- **Using fetch**:

```
useEffect(() => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => setData(data))
    .catch(error => console.error('Error fetching data:', error));
}, []);
```

- **Using Axios**:

```
useEffect(() => {
  axios.get('https://api.example.com/data')
    .then(response => setData(response.data))
    .catch(error => console.error('Error fetching data:', error)); }, []);
```