**SECTION 1: REACT BASICS**

**1)What is React?**

**Answer:**
React is a **JavaScript library** used to build **user interfaces**, especially **single-page applications (SPAs)**.
It follows a **component-based architecture** and uses a **virtual DOM** for fast updates.

2) **Why React is fast?**

**Answer:**

- Uses **Virtual DOM**
- Updates only the **changed part of UI**
- Uses **diffing algorithm**
- Reduces direct DOM manipulation

3) **What is SPA (Single Page Application)?**
**Answer:**
SPA loads a single HTML page and dynamically updates content without reloading the page.
React applications are SPAs.
Example:

- Gmail
- Facebook
- LinkedIn

**4)What is JSX?**
**Answer:**
JSX is **JavaScript XML**.
It allows writing HTML inside JavaScript.
const element = <h1>Hello React</h1>;
JSX is converted into JavaScript using **Babel**.

**5. Can browser understand JSX directly?**
**Answer:**
No[ JSX is converted into JavaScript by **Babel** before execution.]

**6. What is Virtual DOM?**
**Answer:**
Virtual DOM is a **lightweight copy of the real DOM**.
Flow:
State Change → Virtual DOM → Compare with previous → Update Real DOM

**7. Difference between Virtual DOM and Real DOM?**

| **Real DOM** | **Virtual DOM** |
|---|---|
| Slow | Fast |
| Updates entire UI | Updates only changed nodes |
| Browser dependent | JavaScript object |

**8. What are Components?**
**Answer:**
Components are **reusable UI blocks**.
Example:

- Header
- Footer
- Navbar
- LoginForm

**9. Types of Components?**
**Answer:**

1. Functional Components (Recommended)
2. Class Components (Old)

## 10. Functional Component Example

```
function Welcome() {
  return <h1>Welcome to React</h1>;
}
export default Welcome;
```

Class Component

```
import React, { Component } from 'react';
class Welcome extends Component {
 constructor(props) {
  super(props);
  this.state = {
   message: 'Hello from a Class Component!'
  };
 }
 changeMessage = () => {
  this.setState({ message: 'You clicked the
button!'});
 };
```

```
 render() {
  return (
   <div>
    <h1>{this.state.message}</h1>
    <p>Welcome, {this.props.name}!</p>
    <button onClick={this.changeMessage}>Click
Me</button>
   </div>
  );
 }
}

export default Welcome;
```

**SECTION 2: PROPS & STATE (MOST IMPORTANT)**

**11. What are Props?**

**Answer:**

Props are **read-only data** passed from parent to child.

```
function Child(props) {
 return <h2>{props.name}</h2>;
}
```

**12. Are props mutable?**

**Answer:** No

Props are **immutable**.

**13. What is State?**

**Answer:** State is a **local, mutable data** managed inside a component.

**14. Difference between Props and State?**

| Props | State |
|---|---|
| Passed from parent | Managed within component |
| Read-only | Mutable |
| Used for communication | Used for UI changes |

**15. useState Hook**

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

useState is a **React Hook** that lets you add **state variables** to functional components.

**16. Why hooks were introduced?**

**Answer:**

- To use state in functional components
- To avoid complex class lifecycle methods
- To reuse logic easily

**SECTION 3: EVENTS & CONDITIONAL RENDERING**

**17. Handling Events in React**

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

**18. Conditional Rendering Example**

**Answer:** showing different UI elements depending on certain conditions,

**IF** condtion

```
function Greeting({ isLoggedIn }) {
 if (isLoggedIn) {
  return <h1>Welcome back!</h1>;
 }
 return <h1>Please log in.</h1>;
}
```

**Using Ternary Operator**

```
function Greeting({ isLoggedIn }) {
 return (
  <div>
   {isLoggedIn ? <h1>Welcome back!</h1> :
<h1>Please log in.</h1>}
  </div>
 );
}
```

**19. Why arrow functions used in React?**

**Answer:**

- Automatically binds this
- Cleaner syntax

---

## 🟢 SECTION 4: LISTS, KEYS & FORMS

---

**20. Rendering List in React**

items.map(item => <li key={item.id}>{item.name}</li>)

**21. Why key is important?**

Answer:
Keys help React identify elements uniquely and improve performance.

**22. Controlled vs Uncontrolled Components**

| Controlled | Uncontrolled |
|---|---|
| State managed by React | DOM manages state |
| Preferred | Less control |

---

**23. Controlled Input Example**

<input value={name} onChange={e => setName(e.target.value)} />

## 🤓 SECTION 5: HOOKS (VERY IMPORTANT)

---

**24. What is useEffect?**

**Answer:**
Used to handle **side effects** like:

- API calls
- subscriptions
- DOM updates

---

**25. useEffect Syntax**

```
useEffect(() => {
  fetchData();
}, []);
```

---

**26. useEffect dependency array**

- [] → runs once
- [value] → runs when value changes
- no array → runs every render

**27. useEffect vs useState**

| useState | useEffect |
|---|---|
| Stores data | Performs side effects |
| Triggers re-render | Executes logic |

---

## 🟢 SECTION 6: API INTEGRATION (REAL PROJECT QUESTIONS)

---

**28. How do you call REST API in React?**

**Answer:** Using:

- fetch()
- axios

---

**29. Axios Example**

```
axios.get("/api/users")
  .then(res => setUsers(res.data))
```

---

**30. Where should API calls be made?**

**Answer:**
Inside useEffect() to avoid infinite loops.

---

**31. How React connects with Spring Boot?**

**Answer:**

- React → Frontend
- Spring Boot → Backend REST APIs
- Communicate via HTTP (JSON)

## 😜 SECTION 7: ROUTING (IMPORTANT)

---

**32. What is React Router?**

**Answer:**
Library used for **navigation** in React SPA.

---

**33. Route Example**

<Route path="/login" element={<Login />} />

---

**34. useParams Hook**

Used to read URL parameters.

---

## 🟢 SECTION 8: ADVANCED INTERMEDIATE QUESTIONS

---

**35. What is Lifting State Up?**
**Answer:**
Moving shared state to the **nearest common parent**.

---

**36. What is Context API?**
**Answer:**
Used to avoid **prop drilling**.

---

**37. Prop Drilling?**
**Answer:**
Passing props through multiple components unnecessarily.

---

**38. Controlled API error handling**

```
try {
  const res = await axios.get(url);
} catch(error) {
  console.log(error);
}
```

---

**39. Difference between useEffect and componentDidMount?**
**Answer:**
useEffect replaces all lifecycle methods in functional components.

---

**40. React vs Angular?**

| React | Angular |
|---|---|
| Library | Framework |
| Virtual DOM | Real DOM |
| Easy learning | Steep learning |

**REACT CODING INTERVIEW QUESTIONS (Basic → Advanced → Tricky → Real Interview Style)**

---

## 🟢 LEVEL 1: BASIC CODING QUESTIONS (Warm-up)

---

**1️⃣ Create a simple counter**
**❓ Question:**
Create a counter with increment and decrement buttons.
**💻 Code:**

```
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <>
      <h2>{count}</h2>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
    </>
  );
}
```

**🧠 Checks: useState, re-render behavior**

---

**2️⃣ Toggle text on button click**
**❓ Show/Hide text on button click.**

```
const Toggle = () => {
  const [show, setShow] = React.useState(true);

  return (
    <>
      <button onClick={() => setShow(!show)}>Toggle</button>
      {show && <p>Hello React</p>}
    </>
  );
};
```

**🧠 Conditional rendering**

---

**3️⃣ Controlled input field**
**❓ Store input value in state and display it.**

```
function InputBox() {
  const [text, setText] = React.useState("");
```

```
 return (
  <>
   <input onChange={e => setText(e.target.value)}
/>
   <p>{text}</p>
  </>
 );
}
```

🧠 Controlled component

---

4️⃣ Render list with keys

❓ Render list of users.

```
const users = ["A", "B", "C"];

users.map((u, index) => (
 <li key={index}>{u}</li>
));
```

🧠 Keys and rendering lists

---

🟡 LEVEL 2: INTERMEDIATE CODING QUESTIONS

---

5️⃣ Fetch API data using useEffect

❓ Fetch and display users from API.

```
useEffect(() => {

fetch("https://jsonplaceholder.typicode.com/user
s")
  .then(res => res.json())
  .then(data => setUsers(data));
}, []);
```

🧠 Side effects, dependency array

---

6️⃣ Prevent infinite API calls

❓ Why this is wrong?

```
useEffect(() => {
 fetchData();
});
```

✅ Correct

```
useEffect(() => {
 fetchData();
}, []);
```

🧠 Dependency understanding

---

7️⃣ Parent to child data flow

❓ Pass name from parent to child.

```
function Parent() {
 return <Child name="React" />;
}

function Child({ name }) {
 return <h1>{name}</h1>;
}
```

🧠 Props

---

8️⃣ Child to parent communication

❓ Update parent state from child.

```
function Parent() {
 const [count, setCount] = useState(0);
 return <Child update={setCount} />;
}

function Child({ update }) {
 return <button onClick={() => update(c => c +
1)}>+</button>;
}
```

🧠 Callback props

---

9️⃣ Form submit handling

❓ Prevent page reload.

```
const handleSubmit = (e) => {
 e.preventDefault();
 console.log("Submitted");
};
```

🧠 Event handling

---

🔟 Conditional rendering using role

```
{role === "admin" && <AdminPanel />}
```

🧠 Logical rendering

---

🟠 LEVEL 3: ADVANCED CODING QUESTIONS

---

1️⃣1️⃣ Search filter in React

❓ Filter list based on input.

```
const filtered = users.filter(u =>
 u.toLowerCase().includes(search.toLowerCase())
);
```

🧠 State + derived data

---

1️⃣2️⃣ Debounce input

❓ Delay API call.

```
useEffect(() => {
 const timer = setTimeout(() => fetchData(search), 500);
 return () => clearTimeout(timer);
}, [search]);
```

🧠 Cleanup, performance

---

1️⃣3️⃣ Custom Hook

❓ Create useFetch.

```
function useFetch(url) {
 const [data, setData] = useState(null);

 useEffect(() => {
  fetch(url).then(res => res.json()).then(setData);
 }, [url]);

 return data;
}
```

🧠 Reusability

---

1️⃣4️⃣ useRef example

❓ Focus input on button click.

```
const inputRef = useRef();

<button onClick={() => inputRef.current.focus()}>
```

🧠 DOM access

---

1️⃣5️⃣ Memoization using React.memo

```
const Child = React.memo(({ value }) => {
 console.log("Rendered");
 return <h3>{value}</h3>;
});
```

🧠 Performance optimization

---

🔴 LEVEL 4: TRICKY REACT QUESTIONS (VERY IMPORTANT)

---

1️⃣6️⃣ Why this count not increment twice?

```
setCount(count + 1);
setCount(count + 1);
```

✅ Correct

```
setCount(prev => prev + 1);
setCount(prev => prev + 1);
```

🧠 State batching

---

1️⃣7️⃣ Why useEffect runs twice in development?

✓ Because of React StrictMode

🧠 React internals

---

1️⃣8️⃣ Fix stale state bug

```
setTimeout(() => {
 setCount(count + 1);
}, 1000);
```

✅

```
setCount(prev => prev + 1);
```

🧠 Closures

---

1️⃣9️⃣ Infinite re-render bug

❓ What's wrong?

```
setCount(count + 1);
```

Inside component body.

🧠 Render cycle understanding

---

2️⃣0️⃣ Why hooks must not be conditional?

```
if (x) useState();
```

❌ Breaks hook order

---

🟣 LEVEL 5: REAL INTERVIEW SCENARIOS (PROJECT STYLE)

---

2️⃣1️⃣ Protected Route

```
return isAuth ? <Outlet /> : <Navigate to="/login" />;
```

🧠 Routing logic

---

2️⃣2️⃣ Role-based UI

```
{user.role === "ADMIN" && <Admin />}
```

🧠 Authorization

---

2️⃣3️⃣ Optimizing large list rendering

✓ Pagination

✓ Lazy loading

✓ Proper keys

---

2️⃣4️⃣ React + Spring Boot API handling

❓ How do you handle CORS?

✓ Enable CORS in backend

✓ Proxy in React

---

2️⃣5️⃣ Error Boundary (Concept + Code)

```
class ErrorBoundary extends React.Component {
 componentDidCatch(error) {
  console.log(error);
 }
}
```

🧠 Error handling

1️⃣ Is setState / setCount synchronous?

❌ No, it is asynchronous

✅ Explanation:

React batches multiple state updates to improve performance.

So state updates don't happen immediately.

❌ Wrong assumption

```
setCount(count + 1);
console.log(count); // old value
```

✅ Correct way

```
setCount(prev => prev + 1);
```

🧠 Interview line:

React batches state updates, so setState is asynchronous.

---

2️⃣ Can we use state without re-render?

❌ No

✅ Explanation:

State exists only to trigger re-renders.

Whenever state changes → component re-renders.

If you want NO re-render?

✓ Use useRef

```
const countRef = useRef(0);
countRef.current++;
```

🧠 Interview line:

State always causes re-render; refs don't.

---

3️⃣ Does React re-render child if parent re-renders?

✓ Yes (by default)

❌ Example

```
function Parent() {
 return <Child />;
}
```

Parent re-render ⇒ Child re-render

✅ How to stop it?

```
const Child = React.memo(() => {
 return <h1>Child</h1>;
});
```

🧠 Interview line:

React re-renders children when parent re-renders unless memoized.

---

4️⃣ Why key should NOT be index?

❌ Index keys cause UI bugs

❌ Problem

```
items.map((item, index) => (
 <li key={index}>{item}</li>
));
```

If list order changes → React confuses elements

✅ Correct

```
<li key={item.id}>{item.name}</li>
```

🧠 Interview line:

Index keys break reconciliation when list changes dynamically.

---

5️⃣ Difference between useEffect & useLayoutEffect

| useEffect | useLayoutEffect |
| --- | --- |
| Runs after paint | Runs before paint |
| Async | Sync |
| No UI blocking | Blocks rendering |
| Used for API calls | Used for DOM measurements |

Example:

```
useLayoutEffect(() => {
 divRef.current.style.color = "red";
});
```

🧠 Interview line:

useLayoutEffect runs before browser paint and can block UI.

---

6️⃣ Can we call hooks inside loops or conditions?

❌ No

❌ Wrong

```
if (x) {
 useState();
}
```

✅ Reason:

Hooks depend on call order.

Conditional usage breaks order → runtime error.

🧠 Interview line:

Hooks must be called at the top level to maintain consistent order.

---

7️⃣ Why state updates are asynchronous?

✅ Explanation:

- Improves performance
- Prevents unnecessary re-renders
- Enables batching

Example:

```
setCount(count + 1);
setCount(count + 1);
```

Result → +1 only

✅ Correct

```
setCount(prev => prev + 1);
setCount(prev => prev + 1);
```

🧠 Interview line:

React batches updates, making state updates asynchronous.

---

8️⃣ Can we stop re-render?

✓ Yes

Methods:

1. React.memo
2. useCallback
3. useMemo

Example:

```
const Child = React.memo(ChildComponent);
```

🧠 Interview line:

Re-renders can be prevented using memoization techniques.

---

9️⃣ Difference between useRef & useState

| useState | useRef |
| --- | --- |
| Causes re-render | No re-render |
| For UI updates | For storing values |
| Async | Sync |

Example:

```
const ref = useRef(0);
ref.current++;
```

🧠 Interview line:

Refs persist values without triggering re-render.

---

🔟 Context API vs Redux (VERY IMPORTANT)

Context API

✓ Small apps

✓ Auth, theme

✓ Avoid prop drilling

```
<UserContext.Provider value={user}>
```

Redux

✓ Large apps

✓ Complex global state

✓ Debugging & middleware

🧠 Interview line:

Context for simple global state, Redux for large scalable apps.

## 🔥 ONE-LINE ANSWERS (INTERVIEW GOLD)

| Question | One-line Answer |
|---|---|
| **Is setState sync?** | No, it's async |
| **State without render?** | No |
| **Child re-render?** | Yes |
| **Index as key?** | Causes bugs |
| **useEffect vs layout?** | After vs before paint |
| **Hooks in loops?** | No |
| **Why async state?** | Batching |
| **Stop render?** | memo |
| **Ref vs state?** | No render |
| **Context vs Redux?** | Small vs large |

**PART 1: REACT OUTPUT PREDICTION QUESTIONS**
**(*Most common trick round in interviews*)**

---

**Q 1️⃣ What is the output?**

```
function App() {
 const [count, setCount] = React.useState(0);

 return (
  <button onClick={() => {
   setCount(count + 1);
   setCount(count + 1);
  }}>
   {count}
  </button>
 );
}
```

✅ Output after one click:

1

🧠 **Why?**

React batches state updates → both use old count.

---

**Q 2️⃣ What is printed?**

```
console.log("A");

useEffect(() => {
 console.log("B");
}, []);
console.log("C");
```

✅ **Output:**

**A**
**C**
**B**

🧠 **Why?**

**useEffect runs after render.**

---

**Q 3️⃣ What is the output?**

```
function App() {
 const [count, setCount] = React.useState(0);

 React.useEffect(() => {
  setCount(count + 1);
 }, []);

 return <h1>{count}</h1>;
}
```

✅ **Output:**

**1**

🧠 **Why?**

**Effect runs once after first render.**

---

**Q 4️⃣ StrictMode question**

```
useEffect(() => {
 console.log("Hello");
}, []);
```

✅ **Output in development:**

**Hello**
**Hello**

🧠 **Why?**

**React StrictMode intentionally runs effects twice.**

---

**Q 5️⃣ Closure trap**

```
useEffect(() => {
 setTimeout(() => {
  console.log(count);
 }, 1000);
}, []);
```

✅ **Output:**

0

🧠 Why?

Closure captures initial state.

---

👷 PART 2: MACHINE CODING ROUND (2-HOUR TASK)

TASK 1: CRUD App (VERY COMMON)

Requirements:

✓ Fetch users from API

✓ Add new user

✓ Delete user

✓ Search user

✓ Loading & error handling

Core Logic (Interview-level)

```
const [users, setUsers] = useState([]);
const [search, setSearch] = useState("");

const filteredUsers = users.filter(u =>

u.name.toLowerCase().includes(search.toLowerCase())
);
```

What interviewer checks:

✓ State management

✓ useEffect usage

✓ Controlled forms

✓ Performance

---

TASK 2: Pagination

```
const currentUsers = users.slice(start, end);
```

🧠 Checks understanding of derived state.

---

TASK 3: Debounced Search

```
useEffect(() => {
 const timer = setTimeout(() => searchAPI(query), 500);
 return () => clearTimeout(timer);
}, [query]);
```

🧠 Checks cleanup & optimization.

---

TASK 4: Authentication UI

✓ Login page

✓ Protected route

✓ Logout

```
return isAuth ? <Dashboard /> : <Login />;
```

---

🐞 PART 3: REACT DEBUGGING INTERVIEW QUESTIONS

---

Bug 1️⃣ Infinite loop

```
useEffect(() => {
 setCount(count + 1);
}, [count]);
```

❌ Problem:

Infinite re-render

✅ Fix:

```
useEffect(() => {
 setCount(prev => prev + 1);
}, []);
```

---

Bug 2️⃣ Input not typing

```
<input value={name} />
```

❌ Missing:

```
onChange={e => setName(e.target.value)}
```

---

Bug 3️⃣ List not updating after delete

❌ Mutating state directly

```
users.splice(index, 1);
```

✅ Correct

```
setUsers(users.filter(u => u.id !== id));
```

---

Bug 4️⃣ Child re-render issue

❌ No memoization

✅ Fix

```
const Child = React.memo(Child);
```

---

Bug 5️⃣ API called multiple times

❌ Missing dependency array

```
useEffect(() => fetchData());
```