

React JS Interview Questions

I am working in MNC, but not in project. I want to switch with three years experience in react.js. what kind of coding interview questions can i expect in companies like infosys,tcs,wipro? Give question and code in single component for each question in details.

That's a great goal! With **3 years of React.js experience**, you're in a good position to switch roles, even to service-based companies like **Infosys, TCS, and Wipro**.

These companies typically ask questions in a **mix of practical React coding + JavaScript fundamentals + problem-solving (DSA basics)**.

I'll provide you with **detailed coding questions**, each in the form of a **single React component**, as you asked.

✓ Question 1: Create a Counter with useState and useEffect

Asked to test hooks understanding.

✓ Requirements:

- A button to increase count
- Show "High count!" when count > 5 using `useEffect`

jsx

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

const Counter = () => {
  const [count, setCount] = useState(0);
  const [message, setMessage] = useState('');

  useEffect(() => {
    if (count > 5) {
      setMessage('High count!');
    } else {
      setMessage('');
    }
  })
}
```

```

    }, [count]);

    return (
      <div className="p-4">
        <h2 className="text-xl mb-2">Count: {count}</h2>
        <button onClick={() => setCount(count + 1)} className="bg-blue-500 text-white
px-4 py-2 rounded">
          Increment
        </button>
        {message} && <p className="mt-2 text-green-600">{message}</p>
      </div>
    );
  };
};

export default Counter;

```

Question 2: Filter a List Based on Search Input

Tests state handling, event binding, and array filtering.

```

jsx

import React, { useState } from 'react';

const SearchList = () => {
  const [searchTerm, setSearchTerm] = useState('');
  const users = ['Alice', 'Bob', 'Charlie', 'David', 'Eve'];

  const filtered = users.filter(user =>
    user.toLowerCase().includes(searchTerm.toLowerCase())
  );

  return (
    <div className="p-4">
      <input
        type="text"
        placeholder="Search users..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}

```

```

        className="border px-2 py-1 mb-2 w-full"
      />
      <ul>
        {filtered.map((user, idx) => (
          <li key={idx} className="border-b py-1">{user}</li>
        ))}
      </ul>
    </div>
  );
};

export default SearchList;

```

✓ Question 3: Build a Todo App (Add + Delete)

Tests list handling, conditional rendering, and immutability.

```

jsx

import React, { useState } from 'react';

const TodoApp = () => {
  const [task, setTask] = useState('');
  const [todos, setTodos] = useState([]);

  const addTask = () => {
    if (task.trim()) {
      setTodos([...todos, task]);
      setTask('');
    }
  };

  const deleteTask = (index) => {
    const updated = todos.filter((_, i) => i !== index);
    setTodos(updated);
  };

  return (
    <div className="p-4">

```

```

<h2 className="text-xl mb-2">Todo App</h2>
<div className="mb-2">
  <input
    value={task}
    onChange={(e) => setTask(e.target.value)}
    placeholder="Enter task"
    className="border px-2 py-1 mr-2"
  />
  <button onClick={addTask} className="bg-green-500 text-white px-3 py-1
rounded">
    Add
  </button>
</div>
<ul>
  {todos.map((t, i) => (
    <li key={i} className="flex justify-between items-center border-b py-1">
      {t}
      <button onClick={() => deleteTask(i)} className="text-red-
500">Delete</button>
    </li>
  ))}
</ul>
</div>
);
};

export default TodoApp;

```

✓ Question 4: Toggle Component with useState

Simple toggle UI, used to test React basics.

```

jsx

import React, { useState } from 'react';

const ToggleText = () => {
  const [show, setShow] = useState(true);

```

```

return (
  <div className="p-4">
    <button onClick={() => setShow(!show)} className="bg-purple-600 text-white px-
4 py-2 rounded">
      Toggle
    </button>
    {show && <p className="mt-2 text-gray-700">Now you see me!</p>}
  </div>
);
};

export default ToggleText;

```

✓ Question 5: Controlled Form Component

Tests form control, state updates, and submission logic.

```

jsx

import React, { useState } from 'react';

const FormExample = () => {
  const [form, setForm] = useState({ name: '', email: '' });

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name: ${form.name}, Email: ${form.email}`);
  };

  return (
    <form onSubmit={handleSubmit} className="p-4 space-y-2">
      <input
        name="name"
        value={form.name}
        onChange={handleChange}

```

```

        placeholder="Name"
        className="border px-2 py-1 w-full"
      />
      <input
        name="email"
        value={form.email}
        onChange={handleChange}
        placeholder="Email"
        className="border px-2 py-1 w-full"
      />
      <button type="submit" className="bg-blue-600 text-white px-4 py-2 rounded">
        Submit
      </button>
    </form>
  );
};

export default FormExample;

```

Do you want me to continue with:

- API handling with `fetch` or `axios`
- React Router
- `useReducer` or Context API
- DSA-type questions using JS (e.g., reverse string, debounce, flatten array)?

Let me know your priority or the type of role you're targeting (frontend dev, fullstack, etc.), and I can tailor the next set accordingly.

add 10 more

Awesome! Here's **10 more React + JavaScript coding questions** — realistic ones that companies like **Infosys, TCS, Wipro** often ask in interviews for 2–4 years of experience.

Each is in a **single React component**, covering key concepts like **API calls, routing, debounce, memoization, state management, hooks, JavaScript logic**, etc.

✓ Question 6: Fetch Data from API with Loading State

Tests `useEffect`, `fetch` API, and conditional rendering.

jsx

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

const UserList = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then((res) => res.json())
      .then((data) => {
        setUsers(data);
        setLoading(false);
      });
  }, []);

  return (
    <div className="p-4">
      <h2 className="text-xl mb-2">User List</h2>
      {loading ? <p>Loading...</p> : (
        <ul>
          {users.map((user) => (
            <li key={user.id} className="border-b py-1">{user.name}</li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default UserList;
```

✓ Question 7: Debounce Search Input (useEffect + setTimeout)

Tests performance pattern knowledge (debounce).

jsx

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

const DebouncedInput = () => {
  const [search, setSearch] = useState('');
  const [debounced, setDebounced] = useState('');

  useEffect(() => {
    const timer = setTimeout(() => {
      setDebounced(search);
    }, 500); // 500ms debounce

    return () => clearTimeout(timer);
  }, [search]);

  return (
    <div className="p-4">
      <input
        value={search}
        onChange={(e) => setSearch(e.target.value)}
        placeholder="Type here..."
        className="border px-2 py-1"
      />
      <p className="mt-2">Debounced: {debounced}</p>
    </div>
  );
};

export default DebouncedInput;
```

✅ Question 8: Memoize Expensive Calculation

Tests useMemo + performance optimization

jsx


```

import React, { useState, useMemo } from 'react';

const ExpensiveCalc = () => {
  const [num, setNum] = useState(1);

  const factorial = useMemo(() => {
    let result = 1;
    for (let i = 1; i <= num; i++) {
      result *= i;
    }
    console.log("Calculated!");
    return result;
  }, [num]);

  return (
    <div className="p-4">
      <input
        type="number"
        value={num}
        onChange={(e) => setNum(Number(e.target.value))}
        className="border px-2 py-1"
      />
      <p className="mt-2">Factorial: {factorial}</p>
    </div>
  );
};

export default ExpensiveCalc;

```



Question 9: React Router: Navigation Between Pages

Tests knowledge of routing in React.

jsx

```

// Install react-router-dom first
// npm install react-router-dom

import React from 'react';

```

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

const Home = () => <h2 className="text-xl">Home Page</h2>;
const About = () => <h2 className="text-xl">About Page</h2>;

const RouterExample = () => {
  return (
    <BrowserRouter>
      <nav className="space-x-4 p-4 bg-gray-100">
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
};

export default RouterExample;
```

✓ Question 10: Shared State Using Context API

Tests global state without Redux.

```
jsx

import React, { useContext, useState, createContext } from 'react';

const ThemeContext = createContext();

const Child = () => {
  const { theme, toggle } = useContext(ThemeContext);
  return (
    <div>
      <p>Current theme: {theme}</p>
      <button onClick={toggle} className="bg-gray-700 text-white px-2 py-1 rounded">Toggle Theme</button>
    </div>
  );
};
```

```

    </div>
  );
};

const ContextExample = () => {
  const [theme, setTheme] = useState('light');
  const toggle = () => setTheme(theme === 'light' ? 'dark' : 'light');

  return (
    <ThemeContext.Provider value={{ theme, toggle }}>
      <div className="p-4">
        <Child />
      </div>
    </ThemeContext.Provider>
  );
};

export default ContextExample;

```

✓ Question 11: Reverse a String in React

Tests JS string logic with state/UI binding.

```

jsx

import React, { useState } from 'react';

const ReverseString = () => {
  const [text, setText] = useState('');

  const reversed = text.split('').reverse().join('');

  return (
    <div className="p-4">
      <input
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Enter text"
        className="border px-2 py-1"

```

```

    />
    <p className="mt-2">Reversed: {reversed}</p>
  </div>
);
};

export default ReverseString;

```

✓ Question 12: Flatten a Nested Array

Tests JS array logic + render flat list

jsx

```

import React from 'react';

const FlattenArray = () => {
  const nested = [1, [2, 3], [4, [5, 6]], 7];

  const flatten = (arr) => {
    return arr.reduce(
      (acc, val) => acc.concat(Array.isArray(val) ? flatten(val) : val),
      []
    );
  };

  const flatArray = flatten(nested);

  return (
    <div className="p-4">
      <h2 className="text-xl mb-2">Flattened Array</h2>
      <pre>{JSON.stringify(flatArray)}</pre>
    </div>
  );
};

export default FlattenArray;

```

✓ Question 13: Countdown Timer

Tests useEffect with cleanup, time logic

```
jsx

import React, { useState, useEffect } from 'react';

const Countdown = () => {
  const [time, setTime] = useState(10);

  useEffect(() => {
    if (time <= 0) return;
    const timer = setTimeout(() => setTime(time - 1), 1000);
    return () => clearTimeout(timer);
  }, [time]);

  return (
    <div className="p-4">
      <h2 className="text-xl">Time Left: {time}</h2>
    </div>
  );
};

export default Countdown;
```

✓ Question 14: Simple Tabs Component

Tests conditional rendering + UI logic

```
jsx

import React, { useState } from 'react';

const Tabs = () => {
  const [active, setActive] = useState('Home');

  const tabs = ['Home', 'Profile', 'Settings'];
```

```

return (
  <div className="p-4">
    <div className="flex space-x-2 mb-4">
      {tabs.map(tab => (
        <button
          key={tab}
          onClick={() => setActive(tab)}
          className={`px-4 py-2 ${active === tab ? 'bg-blue-600 text-white' : 'bg-gray-200'} `}
        >
          {tab}
        </button>
      ))}
    </div>
    <div>
      <h2 className="text-xl">Current Tab: {active}</h2>
    </div>
  </div>
);
};

export default Tabs;

```

Question 15: Display Unique Items from Array

Tests **Set** , deduplication logic in JS

jsx

```

import React from 'react';

const UniqueItems = () => {
  const items = ['apple', 'banana', 'apple', 'orange', 'banana'];

  const uniqueItems = [...new Set(items)];

  return (
    <div className="p-4">

```

```
<h2 className="text-xl mb-2">Unique Fruits</h2>
<ul>
  {uniqueItems.map((fruit, i) => (
    <li key={i}>{fruit}</li>
  ))}
</ul>
</div>
);
};

export default UniqueItems;
```

Let me know if you want:

- More **DSA in JS** (palindrome, anagram, etc.)
- More **real-world React projects**
- Or if you're preparing for **product-based companies** instead of service-based ones

I can also bundle these into a downloadable zip/project if needed.

How to add these in app.jsx?

Great question! You can import and use all these components inside your `App.jsx` file.

Here's a **step-by-step guide** to include all the components you've seen above in a clean, switchable layout.

✅ Step 1: Folder Structure (Recommended)

plaintext

```
/src
  /components
    Counter.jsx
    SearchList.jsx
    TodoApp.jsx
```

```
ToggleText.jsx
FormExample.jsx
UserList.jsx
DebouncedInput.jsx
ExpensiveCalc.jsx
RouterExample.jsx
ContextExample.jsx
ReverseString.jsx
FlattenArray.jsx
Countdown.jsx
Tabs.jsx
UniqueItems.jsx
App.jsx
index.js
```

✅ Step 2: App.jsx – Add Component Import and Display Logic

Here's a version of `App.jsx` where you can switch between components using a dropdown:

```
jsx

import React, { useState } from 'react';
import Counter from './components/Counter';
import SearchList from './components/SearchList';
import TodoApp from './components/TodoApp';
import ToggleText from './components/ToggleText';
import FormExample from './components/FormExample';
import UserList from './components/UserList';
import DebouncedInput from './components/DebouncedInput';
import ExpensiveCalc from './components/ExpensiveCalc';
import ContextExample from './components/ContextExample';
import ReverseString from './components/ReverseString';
import FlattenArray from './components/FlattenArray';
import Countdown from './components/Countdown';
import Tabs from './components/Tabs';
import UniqueItems from './components/UniqueItems';

const componentMap = {
```



```

Counter,
SearchList,
TodoApp,
ToggleText,
FormExample,
UserList,
DebounceInput,
ExpensiveCalc,
ContextExample,
ReverseString,
FlattenArray,
Countdown,
Tabs,
UniqueItems,
};

const App = () => {
  const [selected, setSelected] = useState('Counter');
  const SelectedComponent = componentMap[selected];

  return (
    <div className="p-6">
      <h1 className="text-2xl font-bold mb-4">React Practice Playground</h1>

      <select
        value={selected}
        onChange={(e) => setSelected(e.target.value)}
        className="border px-3 py-2 mb-4"
      >
        {Object.keys(componentMap).map((key) => (
          <option key={key} value={key}>{key}</option>
        ))}
      </select>

      <div className="mt-4 p-4 border rounded shadow-md">
        <SelectedComponent />
      </div>
    </div>
  );
};

export default App;

```

✓ Step 3: `index.js` – Ensure `App` is rendered

jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './index.css'; // optional, if you're using Tailwind or global styles

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

✓ Step 4 (Optional): Add TailwindCSS for Styling

If you like how those components look (like buttons with colors, padding, etc.), install and setup **TailwindCSS** by following:

bash

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Then in `tailwind.config.js`, add:

js

```
content: ['./src/**/*.{js,jsx,ts,tsx}'],
```

And in `index.css`:

css

```
@tailwind base;
@tailwind components;
```

```
@tailwind utilities;
```

Want me to generate the full zip of this ready-to-run React project with all files?

Or should I generate a GitHub-ready version for you?

Do you like this personality