75 React Interview Questions with Answers

Fundamentals

1. What is React?

Answer: React is a JavaScript library for building user interfaces, particularly single-page applications. It's maintained by Meta (formerly Facebook) and focuses on component-based architecture.

Frequency: Very High (95%)
Companies: Facebook, Amazon, Netflix, Airbnb, Uber

2. What are the key features of React?

Answer:

- Virtual DOM for performance optimization
- Component-based architecture
- Unidirectional data flow
- JSX syntax
- React Hooks
- Server-side rendering capabilities

Frequency: High (85%)
Companies: Microsoft, Google, Twitter, PayPal

3. What is JSX?

Answer: JSX is a syntax extension for JavaScript that looks similar to HTML. It allows you to write HTML-like code in JavaScript files, which gets transformed into React.createElement() calls during the build process.

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

// Compiles to
const element = React.createElement('h1', null, 'Hello, world!');
```

Frequency: Very High (90%)
Companies: Facebook, Dropbox, Pinterest, Walmart

4. Why can't browsers read JSX directly?

Answer: Browsers can only read JavaScript objects. JSX is not valid JavaScript; it's a syntax extension that needs to be transformed into regular JavaScript using tools like Babel before it can be understood by browsers.

Frequency: Medium (60%)
Companies: Facebook, Slack, Salesforce

5. What is the difference between Element and Component in React?

Answer:

- React Element: A plain object describing what you want to see on the screen. It's a lightweight description of the DOM node.
- React Component: A function or class that returns React elements. Components can have state, props, and lifecycle methods.

Frequency: Medium (65%)
Companies: Amazon, Uber, Microsoft

6. What is the Virtual DOM?

Answer: Virtual DOM is a lightweight JavaScript representation of the actual DOM. React uses it to improve performance by minimizing direct DOM manipulations. When state changes, React creates a new Virtual DOM tree, compares it with the previous one (diffing), and then only updates the real DOM with the necessary changes.

Frequency: Very High (90%)
Companies: Facebook, Netflix, Airbnb, Spotify

7. What are Controlled Components?

Answer: Controlled components are form elements whose values are controlled by React state. The form data is handled by the React component rather than the DOM, making the React state the "single source of truth."

```
function ControlledInput() {
  const [value, setValue] = useState('');

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

Frequency: High (80%)
Companies: Google, Stripe, Facebook, Shopify

8. What are Uncontrolled Components?

Answer: Uncontrolled components are form elements that maintain their own state internally. Data is retrieved from the DOM using refs rather than through event handlers.

```
function UncontrolledInput() {
  const inputRef = useRef(null);

  function handleSubmit() {
    console.log(inputRef.current.value);
  }

  return (
    <>
      <input ref={inputRef} defaultValue="Default value" />
      <button onClick={handleSubmit}>Submit</button>
    </>
  );
}
```

Frequency: Medium (60%)
Companies: LinkedIn, Twitter, Atlassian

9. What is the difference between state and props?

Answer:

- Props (short for properties): Read-only data passed from parent to child components. They cannot be modified by the component receiving them.
- State: Managed within a component and can be updated using setState() or state updater functions (with hooks). Changes to state trigger re-renders.

Frequency: Very High (95%)
Companies: Facebook, Amazon, Netflix, Google, Microsoft

10. What are Pure Components?

Answer: Pure Components are similar to regular components but implement shouldComponentUpdate() with a shallow comparison of props and state. They only re-render if props or state have changed, which can improve performance by avoiding unnecessary renders.

Frequency: Medium (65%)
Companies: Facebook, Uber, LinkedIn, Airbnb

React Hooks

11. What are React Hooks?

Answer: Hooks are functions that let you "hook into" React state and lifecycle features from functional components. They were introduced in React 16.8 to allow using state and other React features without writing classes.

Frequency: Very High (90%)
Companies: Facebook, Airbnb, Stripe, Uber, Twitter

12. Explain useState Hook.

Answer: useState is a Hook that lets you add state to functional components. It returns an array with two elements: the current state value and a function to update it.

```
function Counter() {
  // Destructuring the array returned by useState
  const [count, setCount] = useState(0); // Initial state is 0

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Frequency: Very High (95%)
Companies: Facebook, Amazon, Netflix, Google, almost all React-based companies

13. Explain useEffect Hook.

Answer: useEffect lets you perform side effects in functional components. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount combined in class components.

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

  useEffect(() => {
    // This runs after render and when dependencies change
    fetchData().then(result => setData(result));

    // Cleanup function (equivalent to componentWillUnmount)
    return () => {
      // Cleanup code
    };
  }, [/* dependencies array */]); // Empty array means run only on mount and unmount

  return <div>{data}</div>;
}
```

Frequency: Very High (95%)
Companies: Facebook, Amazon, Netflix, Google, almost all React-based companies

14. What is useContext Hook?

Answer: useContext is a Hook that accepts a context object (created by React.createContext) and returns the current context value. It's used for consuming context in functional components without wrapper components.

```
// Create context
const ThemeContext = React.createContext('light');

function ThemedButton() {
  // Use context
  const theme = useContext(ThemeContext);
  return <button className={theme}>Themed Button</button>;
}
```

Frequency: High (75%)
Companies: Facebook, Twitter, Shopify, Atlassian

15. Explain useReducer Hook.

Answer: useReducer is a Hook for state management that's preferable to useState when you have complex state logic. It accepts a reducer function and initial state, returning the current state and a dispatch function.

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, {count: 0});

  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
    </>
  );
}
```

Frequency: Medium-High (70%)

Companies: Facebook, Slack, Stripe, GitHub

16. What is useCallback Hook?

Answer: useCallback returns a memoized version of a callback function that only changes if one of the dependencies has changed. It's useful for optimizing child component renders when passing callbacks to them.

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

  // This function is recreated only when count changes
  const handleClick = useCallback(() => {
    console.log(`Button clicked, count: ${count}`);
  }, [count]);

  return <ChildComponent onClick={handleClick} />;
}
```

Frequency: Medium-High (70%)
Companies: Facebook, Uber, Airbnb, Twitter

17. What is useMemo Hook?

Answer: useMemo returns a memoized value that's only recomputed when one of its dependencies changes. It's used to avoid expensive calculations on every render.

```
function ExpensiveComponent({ list, filter }) {
  // filteredList is recomputed only when list or filter changes
  const filteredList = useMemo(() => {
    return list.filter(item => item.includes(filter));
  }, [list, filter]);

  return (
    <div>
      {filteredList.map(item => <div key={item}>{item}</div>)}
    </div>
  );
}
```

Frequency: Medium-High (70%)
Companies: Facebook, Netflix, Dropbox, Stripe

18. What is useRef Hook?

Answer: useRef returns a mutable ref object with a .current property initialized to the passed argument. It persists across renders and doesn't cause re-renders when its value changes.

```jsx
function TextInputWithFocusButton() {
  const inputRef = useRef(null);

  function handleClick() {
    // Directly access the DOM element
    inputRef.current.focus();
  }

  return (
    <>
      <input ref={inputRef} type="text" />
      <button onClick={handleClick}>Focus the input</button>
    </>
  );
}
```

Frequency: High (75%)
Companies: Facebook, Amazon, Airbnb, Spotify

19. What is useLayoutEffect Hook?

Answer: useLayoutEffect is similar to useEffect but fires synchronously after all DOM mutations. Use it when you need to read layout from the DOM and synchronously re-render, while useEffect is deferred until after the browser has painted.

Frequency: Medium (50%)
Companies: Facebook, Twitter, Shopify, Atlassian

20. What is useImperativeHandle Hook?

Answer: useImperativeHandle customizes the instance value that is exposed when using ref with forwardRef. It lets a parent component access specific functions or properties from a child component.

```jsx
const ChildComponent = forwardRef((props, ref) => {
  const inputRef = useRef();

  useImperativeHandle(ref, () => ({
    focus: () => {
      inputRef.current.focus();
    }
  }));

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

function ParentComponent() {
  const childRef = useRef();

  function handleClick() {
    childRef.current.focus();
  }

  return (
    <>
      <ChildComponent ref={childRef} />
      <button onClick={handleClick}>Focus Child Input</button>
    </>
  );
}
```

Frequency: Low (30%)
Companies: Facebook, Airbnb, Dropbox

Advanced Concepts

21. What are Higher-Order Components (HOCs)?

Answer: HOCs are functions that take a component and return a new component with enhanced functionality. They're used for code reuse, logic abstraction, and cross-cutting concerns like authentication or data fetching.

```
// HOC example
function withLogging(WrappedComponent) {
  return function WithLogging(props) {
    useEffect(() => {
      console.log(`Component ${WrappedComponent.name} mounted`);
      return () => {
        console.log(`Component ${WrappedComponent.name} unmounted`);
      };
    }, []);

    return <WrappedComponent {...props} />;
  };
}

// Usage
const EnhancedComponent = withLogging(MyComponent);
```

Frequency: Medium-High (65%)
Companies: Facebook, Netflix, LinkedIn, Uber

22. What is React.memo() and when should you use it?

Answer: React.memo() is a higher-order component that memoizes a component, preventing unnecessary re-renders if props haven't changed. Use it for pure functional components that render the same result given the same props.

```
const MyComponent = React.memo(function MyComponent(props) {
  /* render using props */
});
```

Frequency: Medium-High (70%)
Companies: Facebook, Amazon, Uber, Twitter

23. What are render props?

Answer: Render Props is a technique for sharing code between components using a prop whose value is a function. The component with the render prop calls this function with its state and renders the result.

```
function MouseTracker() {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  function handleMouseMove(event) {
    setPosition({
      x: event.clientX,
      y: event.clientY
    });
  }

  return (
    <div onMouseMove={handleMouseMove}>
      {/* The "render prop" function */}
      {props.render(position)}
    </div>
  );
}

// Usage
<MouseTracker
  render={position => (
    <p>Mouse position: {position.x}, {position.y}</p>
  )}
/>
```

Frequency: Medium (55%)

Companies: Facebook, LinkedIn, Airbnb, Stripe

24. How does the Context API work?

Answer: Context API provides a way to share values like themes, user data, or localization between components without explicitly passing props through every level of the component tree.

```jsx
// Create context with default value
const ThemeContext = React.createContext('light');

// Provider in parent component
function App() {
  const [theme, setTheme] = useState('dark');

  return (
    <ThemeContext.Provider value={theme}>
      <Layout />
      <button onClick={() => setTheme(theme === 'dark' ? 'light' : 'dark')}>
        Toggle Theme
      </button>
    </ThemeContext.Provider>
  );
}

// Consumer in deeply nested component
function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>Themed Button</button>;
}
```

Frequency: High (80%)

Companies: Facebook, Twitter, Airbnb, Shopify

25. What is code-splitting in React?

Answer: Code-splitting is a technique to split your code into smaller chunks that can be loaded on demand, improving application load time. React supports code-splitting via dynamic imports and React.lazy.

```
// Without code-splitting
import BigComponent from './BigComponent';

// With code-splitting
const BigComponent = React.lazy(() => import('./BigComponent'));

function MyComponent() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <BigComponent />
    </React.Suspense>
  );
}
```

Frequency: Medium (60%)
Companies: Facebook, Netflix, Amazon, Uber

26. What is React.lazy() and Suspense?

Answer: React.lazy() lets you render a dynamic import as a regular component. Suspense lets you specify a loading indicator while waiting for lazy-loaded components to load.

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

function MyComponent() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <OtherComponent />
    </React.Suspense>
  );
}
```

Frequency: Medium-High (65%)
Companies: Facebook, Netflix, Airbnb, Spotify

27. What are Error Boundaries in React?

Answer: Error Boundaries are React components that catch JavaScript errors in their child component tree, log those errors, and display a fallback UI. They catch errors during rendering, in lifecycle methods, and in constructors.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state to render fallback UI
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // Log the error
    console.error("Caught an error:", error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}

// Usage
<ErrorBoundary>
  <MyComponent />
</ErrorBoundary>
```

Frequency: Medium (55%)
Companies: Facebook, Netflix, Shopify, Atlassian

28. What is prop drilling and how can you avoid it?

Answer: Prop drilling is the process of passing props through multiple levels of components that don't need those props but only pass them down. To avoid it, you can use Context API, composition, or state management libraries like Redux.

Frequency: Medium-High (70%)
Companies: Facebook, Uber, Twitter, Stripe

29. What is the StrictMode in React?

Answer: StrictMode is a tool for highlighting potential problems in an application. It activates additional checks and warnings for its descendants and doesn't affect production builds.

```
<React.StrictMode>
  <App />
</React.StrictMode>
```

Frequency: Medium (50%)
Companies: Facebook, Twitter, Shopify, Amazon

30. What is React Fiber?

Answer: React Fiber is the new reconciliation algorithm in React 16+. It enables incremental rendering, splitting rendering work into chunks and spreading it over multiple frames, allowing React to pause, abort, or resume work as needed.

Frequency: Medium-Low (40%)
Companies: Facebook, Netflix, Uber

React Performance Optimization

31. How can you optimize performance in React applications?

Answer: React performance can be optimized through:

- Using React.memo for functional components
- Implementing shouldComponentUpdate for class components
- Using PureComponent for classes with simple props/state
- Using useCallback and useMemo hooks to memoize functions and calculations
- Implementing code-splitting with React.lazy and Suspense
- Avoiding unnecessary renders with proper key usage in lists
- Keeping component state local when possible
- Using production builds with minification
- Implementing virtualization for long lists

Frequency: High (80%)
Companies: Facebook, Netflix, Airbnb, Uber, Amazon

32. What is the significance of keys in React lists?

Answer: Keys help React identify which items have changed, added, or removed. They give elements a stable identity across renders, improving the reconciliation process and performance.

```
function List({ items }) {
  return (
    <ul>
      {items.map(item => (
        // Using unique and stable IDs as keys is best practice
        <li key={item.id}>{item.text}</li>
      ))}
    </ul>
  );
}
```

Frequency: High (85%)
Companies: Facebook, Google, Amazon, Twitter, most React-based companies

33. What is memoization in React and how do you use it?

Answer: Memoization is an optimization technique that stores the results of expensive function calls and returns the cached result when the same inputs occur again. In React, you can use React.memo, useMemo, and useCallback for memoization.

Frequency: Medium-High (70%)
Companies: Facebook, Netflix, LinkedIn, Uber

34. How does React handle events?

Answer: React uses synthetic events, which are cross-browser wrappers around native browser events. They have the same interface as native events but work identically across browsers and have some performance benefits through event pooling.

```
function handleClick(event) {
  // event is a synthetic event
  event.preventDefault();
  console.log('Button clicked');
}

return <button onClick={handleClick}>Click Me</button>;
```

Frequency: Medium (60%)
Companies: Facebook, Twitter, Amazon, Shopify

35. How can you prevent unnecessary re-renders in React?

Answer:

- For functional components: Use React.memo
- For class components: Extend PureComponent or implement shouldComponentUpdate
- Use useCallback for event handlers passed to child components
- Use useMemo for expensive calculations
- Keep component state as local as possible
- Avoid creating new objects or arrays directly in render

Frequency: High (80%)
Companies: Facebook, Netflix, Uber, Airbnb, Twitter

React Router

36. What is React Router?

Answer: React Router is a standard library for routing in React applications. It enables navigation among views in a React application, allows browsers to maintain history, and keeps the UI in sync with the URL.

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

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Frequency: High (85%)

Companies: Airbnb, Uber, Twitter, PayPal, most React-based companies

37. What are the different types of routers in React Router?

Answer: The main router types in React Router are:

- BrowserRouter: Uses HTML5 History API (clean URLs like /about)
- HashRouter: Uses URL hash (URLs like /#/about)
- MemoryRouter: Keeps history in memory (for testing or non-browser environments)
- StaticRouter: For server-side rendering

Frequency: Medium (60%)

Companies: Airbnb, Uber, LinkedIn, Twitter

38. How do you handle route parameters in React Router?

Answer: Route parameters are defined with a colon in the path and accessed via the useParams hook (in v6) or match.params (in earlier versions).

```
// Route definition
<Route path="/users/:userId" element={<UserProfile />} />

// Component
function UserProfile() {
  // In React Router v6
  const { userId } = useParams();

  return <div>User ID: {userId}</div>;
}
```

Frequency: Medium-High (70%)
Companies: Airbnb, Twitter, PayPal, Shopify

State Management

39. What is Redux and when should you use it?

Answer: Redux is a predictable state container for JavaScript apps that helps manage application state in a single store. Use Redux when:

- You have complex state that's needed across many components
- The state is updated frequently
- The logic to update the state is complex
- The app has a medium or large-sized codebase with many developers
- You need to track how state is updated over time

Frequency: High (85%)
Companies: Twitter, Instagram, Airbnb, Netflix, many large React applications

40. What are the three principles of Redux?

Answer:

1. Single source of truth: The state is stored in a single store
2. State is read-only: The only way to change state is to dispatch an action
3. Changes are made with pure functions: Reducers are pure functions that take the previous state and an action to return a new state

Frequency: Medium-High (70%)
Companies: Twitter, Airbnb, Netflix, Uber

41. What is Flux architecture?

Answer: Flux is an application architecture pattern developed by Facebook for building client-side web applications. It complements React's composable view components by utilizing a unidirectional data flow. The main components are the Dispatcher, Stores, Views, and Actions.

Frequency: Medium-Low (40%)
Companies: Facebook, early React adopters

42. What is the difference between Redux and React's Context API?

Answer:

- Redux: More powerful for complex state management, includes middleware support, DevTools for debugging, and predictable state updates through reducers.
- Context API: Simpler for less complex applications, built into React, no extra dependencies, but less optimized for frequent updates and lacks middleware capabilities.

Frequency: Medium-High (70%)
Companies: Twitter, Stripe, Shopify, PayPal

43. What is Redux Toolkit and how does it differ from standard Redux?

Answer: Redux Toolkit is the official, opinionated toolset for efficient Redux development. It simplifies Redux code by:

- Including utilities to simplify common use cases
- Providing good defaults for store setup
- Including Redux Thunk for async logic
- Using Immer to enable "mutating" logic in reducers
- Removing boilerplate code with createSlice

```
import { createSlice, configureStore } from '@reduxjs/toolkit';

// Create a slice (reducer + actions)
const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: state => {
      // Can write "mutating" logic thanks to Immer
      state.value += 1;
    },
    decrement: state => {
      state.value -= 1;
    }
  }
});

// Extract action creators and reducer
export const { increment, decrement } = counterSlice.actions;
export const counterReducer = counterSlice.reducer;

// Configure store
const store = configureStore({
  reducer: {
    counter: counterReducer
  }
});
```

Frequency: Medium-High (65%)

Companies: Modern companies using Redux: Airbnb, Twitter, Spotify

44. What are middleware in Redux?

Answer: Middleware in Redux provides a third-party extension point between dispatching an action and the moment it reaches the reducer. It's used for logging, crash reporting, async actions, routing, and more. Redux Thunk is a popular middleware for handling async actions.

Frequency: Medium-High (65%)

Companies: Netflix, Twitter, Uber, Airbnb

45. What is Recoil and how does it differ from Redux?

Answer: Recoil is a state management library developed by Facebook specifically for React. Unlike Redux:

- It uses atoms (sharable state units) instead of a single store
- It has built-in support for derived state through selectors
- It's designed with React's Concurrent Mode in mind
- It has a more React-centric API with hooks
- It's easier to adopt incrementally

Frequency: Low-Medium (30%)
Companies: Facebook, some modern React applications

Testing

46. What are the different types of tests in React applications?

Answer:

- Unit Tests: Testing individual components in isolation
- Integration Tests: Testing interactions between components
- Snapshot Tests: Capturing and comparing component output
- End-to-End Tests: Testing the complete application workflow

Frequency: Medium-High (70%)
Companies: Facebook, Twitter, Airbnb, Netflix, enterprise applications

47. What is Jest?

Answer: Jest is a JavaScript testing framework developed by Facebook. It's frequently used for testing React applications, providing a test runner, assertion library, mocking capabilities, and snapshot testing.

Frequency: High (75%)
Companies: Facebook, Twitter, Airbnb, Netflix, most React-based companies

48. What is React Testing Library?

Answer: React Testing Library is a set of utilities that encourages good testing practices by testing components in a way that resembles how users interact with your app. It focuses on testing behavior rather than implementation details.

```
import { render, screen, fireEvent } from '@testing-library/react';
import Counter from './Counter';

test('increments counter on button click', () => {
  render(<Counter />);

  // Find elements by their accessible text
  const counterValue = screen.getByText(/count: 0/i);
  const incrementButton = screen.getByRole('button', { name: /increment/i });

  // Simulate user interactions
  fireEvent.click(incrementButton);

  // Assert on the expected outcome
  expect(counterValue).toHaveTextContent('Count: 1');
});
```

Frequency: High (80%)
Companies: Twitter, Airbnb, Netflix, most modern React applications

49. What are snapshot tests?

Answer: Snapshot tests capture the rendered output of a component and compare it to a saved "snapshot" from a previous test run. They're useful for detecting unintended UI changes but need careful management to avoid false positives.

```
import renderer from 'react-test-renderer';
import Button from './Button';

test('Button renders correctly', () => {
  const tree = renderer.create(<Button label="Click me" />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Frequency: Medium (60%)

Companies: Facebook, Twitter, Airbnb, most React-based companies

50. How do you test hooks in React?

Answer: React hooks can be tested using the @testing-library/react-hooks package, which provides a renderHook function to test custom hooks in isolation.

```
import { renderHook, act } from '@testing-library/react-hooks';
import useCounter from './useCounter';

test('useCounter increments count', () => {
  const { result } = renderHook(() => useCounter());

  // Initial state
  expect(result.current.count).toBe(0);

  // Update state
  act(() => {
    result.current.increment();
  });

  // Check updated state
  expect(result.current.count).toBe(1);
});
```

Frequency: Medium (55%)

Companies: Twitter, Netflix, modern React applications

React Server Components

51. What are React Server Components?

Answer: React Server Components (RSC) are a new kind of component that runs only on the server. They can access server resources directly (like databases), reduce bundle size by keeping server-only code out of client bundles, and never rerender in the browser.

Frequency: Medium (increasing, 50%)

Companies: Facebook/Meta, Vercel, modern React applications, Next.js users

52. How do React Server Components differ from Client Components?

Answer:

- Server Components: Run only on server, can't use hooks or browser APIs, can directly access backend resources, don't increase bundle size
- Client Components: Run in the browser, can use state/effects, can access browser APIs, contribute to bundle size

```
// Server Component (file.server.jsx or using 'use server')
async function ServerComponent() {
  // Can fetch data directly
  const data = await db.query('SELECT * FROM users');
  return <div>{data.map(user => <p>{user.name}</p>)}</div>;
}

// Client Component (file.client.jsx or using 'use client')
'use client';
function ClientComponent() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}
```

Frequency: Medium (increasing, 50%)
Companies: Meta/Facebook, Vercel, Next.js users

53. What is the difference between server-side rendering (SSR) and React Server Components?

Answer:

- SSR: Renders the entire React component tree on the server to HTML, then hydrates it on the client, making it interactive. All code is still sent to the client.
- Server Components: Allow some components to run exclusively on the server with no client-side JavaScript, while others run on the client. They provide a more granular approach to server/client code splitting.

Frequency: Medium (45%)
Companies: Meta/Facebook, Vercel, modern React applications

React 18 Features

54. What is Concurrent React?

Answer: Concurrent React is a set of features that help React apps stay responsive and adjust to the user's device capabilities and network conditions. It allows React to prepare multiple versions of the UI at the same time, interrupt and defer rendering work, and abandon in-progress renders in favor of more urgent updates.

Frequency: Medium-High (60%)
Companies: Facebook, Netflix, modern React applications

55. What is Automatic Batching in React 18?

Answer: Automatic Batching in React 18 groups multiple state updates into a single re-render for better performance. Unlike React 17, which only batched updates inside React event handlers, React 18 automatically batches all updates regardless of where they come from (event handlers, promises, timeouts, etc.).

```
// In React 17
function handleClick() {
  // These caused two separate renders in React 17
  setCount(c => c + 1);
  setFlag(f => !f);
}

// In React 18
function handleClick() {
  // These are batched into a single render
  setCount(c => c + 1);
  setFlag(f => !f);
}

// Even async updates are batched in React 18
// Note: In React 18, both updates are automatically batched into one render
function handleAsyncClick() {
  fetch('/api/data').then(() => {
    // In React 17, these would cause two renders
    // In React 18, these are automatically batched
    setCount(c => c + 1);
    setFlag(f => !f);
  });
}
```

Automatic batching improves performance by reducing the number of renders and preventing intermediate states. If needed, you can opt out of automatic batching using flushSync() for specific updates that must be processed immediately.

Frequency: Medium-High (65%)
Companies: Facebook, Netflix, Vercel, modern React applications

56. What is the startTransition API in React 18?

Answer: startTransition is a new API that helps differentiate between urgent updates (like typing, clicking) and non-urgent updates (like search results rendering). It lets React pause rendering of less critical updates during user interactions to keep the app responsive.

```
import { startTransition } from 'react';

// Urgent update - happens immediately
setInputValue(input);

// Non-urgent update - can be interrupted
startTransition(() => {
  setSearchResults(searchByQuery(input));
});
```

Frequency: Medium (55%)

Companies: Facebook, Vercel, modern React applications

57. What is useTransition hook in React 18?

Answer: useTransition is a React Hook that returns a stateful value showing whether a transition is pending and a function to start the transition. It helps manage state transitions while keeping the UI responsive.

```jsx
import { useTransition } from 'react';

function SearchComponent() {
  const [isPending, startTransition] = useTransition();
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);

  function handleChange(e) {
    // Urgent: Update the input
    setSearchQuery(e.target.value);

    // Mark results update as non-urgent transition
    startTransition(() => {
      setSearchResults(filterList(e.target.value));
    });
  }

  return (
    <>
      <input value={searchQuery} onChange={handleChange} />
      {isPending ? <p>Loading results...</p> :
        searchResults.map(result => <Item key={result.id} item={result} />)
      }
    </>
  );
}
```

Frequency: Medium (50%)
Companies: Facebook, Vercel, Netflix, modern React applications

58. What is the useDeferredValue hook?

Answer: useDeferredValue accepts a value and returns a new copy of that value that will defer to more urgent updates. It's similar to debouncing or throttling but integrated with React's rendering system.

```jsx
import { useDeferredValue } from 'react';

function SearchResults({ query }) {
  // Defer the expensive re-rendering
  const deferredQuery = useDeferredValue(query);

  // This component re-renders when deferredQuery changes
  // which may lag behind the actual query value
  const results = useMemo(
    () => computeExpensiveResults(deferredQuery),
    [deferredQuery]
  );

  return (
    <div>
      {results.map(result => <Item key={result.id} item={result} />)}
    </div>
  );
}
```

Frequency: Medium (45%)

Companies: Facebook, Vercel, modern React applications

59. What is Suspense in React?

Answer: Suspense is a React feature that lets your components "wait" for something before they render. It handles loading states automatically and can be used with React.lazy for code splitting and data fetching in React 18+.

```jsx
import { Suspense } from 'react';

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

function MyComponent() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

Frequency: Medium-High (65%)

Companies: Facebook, Netflix, Vercel, modern React applications

60. What is the useId hook in React 18?

Answer: useId is a hook for generating unique IDs on both the client and server, while avoiding hydration mismatches. It's useful for accessibility attributes that need unique IDs.

```
import { useId } from 'react';

function NameInput() {
  const id = useId();
  return (
    <>
      <label htmlFor={id}>Name:</label>
      <input id={id} type="text" />
    </>
  );
}
```

Frequency: Low-Medium (35%)

Companies: Facebook, accessibility-focused companies, modern React applications

React Ecosystem & Integration

61. What is Next.js and when would you use it?

Answer: Next.js is a React framework that provides features like server-side rendering, static site generation, API routes, and built-in CSS support. Use it when you need SEO optimization, fast page loads, or a full-stack React application with minimal setup.

Frequency: High (80%)

Companies: Netflix, TikTok, Twitch, Hulu, Uber, many modern React applications

62. What is the difference between CSR, SSR, and SSG in React applications?

Answer:

- CSR (Client-Side Rendering): Initial HTML is minimal, and content is rendered in the browser after JavaScript loads. Good for authenticated applications or dynamic dashboards.
- SSR (Server-Side Rendering): HTML is generated on each request on the server, then hydrated on the client. Good for pages that need SEO and fast FCP.
- SSG (Static Site Generation): HTML is generated at build time and reused for each request. Best for content that doesn't change often.

Frequency: High (75%)
Companies: Netflix, Airbnb, e-commerce companies, content sites

63. What is Styled Components and how does it work?

Answer: Styled Components is a CSS-in-JS library that uses tagged template literals to style components. It generates unique class names for styles, supports dynamic styling based on props, and has theming support.

```
import styled from 'styled-components';

// Create a styled component
const Button = styled.button`
  background: ${props => props.primary ? 'blue' : 'white'};
  color: ${props => props.primary ? 'white' : 'blue'};
  padding: 0.5em 1em;
  border: 2px solid blue;
  border-radius: 3px;
`;

// Use the component
function App() {
  return (
    <div>
      <Button>Normal Button</Button>
      <Button primary>Primary Button</Button>
    </div>
  );
}
```

Frequency: High (75%)
Companies: Airbnb, BBC, Target, Vimeo, many modern React applications

64. What is the React DevTools extension?

Answer: React DevTools is a browser extension that allows you to inspect the React component hierarchy, view component props and state, track component renders, and measure performance with the Profiler. It's essential for debugging and optimizing React applications.

Frequency: Medium-High (70%)
Companies: Most companies using React, especially for performance optimization

65. What are React Portals?

Answer: Portals provide a way to render children into a DOM node outside of their parent component's hierarchy, useful for modals, tooltips, and popups.

```
import { createPortal } from 'react-dom';

function Modal({ children, isOpen }) {
  if (!isOpen) return null;

  return createPortal(
    <div className="modal">
      {children}
      <button onClick={onClose}>Close</button>
    </div>,
    document.getElementById('modal-root') // A DOM node outside your app hierarchy
  );
}
```

Frequency: Medium (60%)
Companies: Airbnb, Netflix, e-commerce sites, applications with modals/overlays

66. How do you style React components?

Answer: React components can be styled using:

- Regular CSS with class names
- Inline styles with the style attribute
- CSS Modules for component-scoped CSS
- CSS-in-JS libraries like Styled Components or Emotion
- Utility-first frameworks like Tailwind CSS

Each approach has trade-offs regarding performance, isolation, and developer experience.

Frequency: High (85%)
Companies: All companies using React

67. What is Framer Motion and how is it used in React?

Answer: Framer Motion is a production-ready motion library for React that powers animations and gestures. It provides a simple declarative syntax for animations, transitions, and gestures.

```
import { motion } from 'framer-motion';

function AnimatedBox() {
  return (
    <motion.div
      initial={{ opacity: 0, scale: 0.5 }}
      animate={{ opacity: 1, scale: 1 }}
      transition={{ duration: 0.5 }}
      whileHover={{ scale: 1.1 }}
      whileTap={{ scale: 0.9 }}
    >
      Animated Box
    </motion.div>
  );
}
```

Frequency: Medium (50%)
Companies: Websites with rich animations, design-focused companies, portfolio sites

68. What is React Query and what problems does it solve?

Answer: React Query is a data-fetching and state management library for React that simplifies fetching, caching, synchronizing, and updating server state. It handles loading states, error states, caching, refetching, and pagination out of the box.

```jsx
import { useQuery } from 'react-query';

function Users() {
  const { isLoading, error, data } = useQuery('users',
    () => fetch('/api/users').then(res => res.json())
  );

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <ul>
      {data.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

Frequency: Medium-High (65%)
Companies: Modern React applications with complex data fetching requirements

69. What is SWR and how does it differ from React Query?

Answer: SWR (stale-while-revalidate) is a React data fetching library that returns cached data first (stale), then fetches new data (revalidate). Like React Query, it handles caching, revalidation, focus tracking, and more, but with a simpler API focused specifically on data fetching.

```jsx
import useSWR from 'swr';

function Profile() {
  const { data, error } = useSWR('/api/user', fetcher);

  if (error) return <div>Failed to load</div>;
  if (!data) return <div>Loading...</div>;

  return <div>Hello {data.name}!</div>;
}
```

Frequency: Medium (45%)
Companies: Vercel, Next.js applications, modern React applications

70. What are React Hooks Rules and why are they important?

Answer: React Hooks Rules are:

1. Only call hooks at the top level (not inside loops, conditions, or nested functions)
2. Only call hooks from React function components or custom hooks

These rules ensure hooks maintain their order between renders, which is crucial for React to correctly preserve state between multiple useState and useEffect calls.

Frequency: High (85%)
Companies: All companies using React with hooks

71. What is the useDebugValue hook?

Answer: useDebugValue is a React Hook for displaying a label for custom hooks in React DevTools. It's useful when creating custom hooks that are shared across an application or as a library.

```
function useCustomFormState(initialState) {
  const [state, setState] = useState(initialState);

  // This will show in React DevTools
  useDebugValue(state === '' ? 'empty' : 'filled');

  return [state, setState];
}
```

Frequency: Low (25%)
Companies: Companies building custom hook libraries, larger React applications

72. What are React's accessibility features?

Answer: React supports building accessible applications through:

- JSX support for all ARIA attributes
- Support for semantic HTML elements
- Managing focus with refs
- Setting lang attributes
- Ensuring keyboard navigation works
- useId hook for generating unique IDs for accessibility attributes

Frequency: Medium (50%)

Companies: Government sites, educational platforms, enterprise applications, companies with legal accessibility requirements

73. What is React.forwardRef and when would you use it?

Answer: React.forwardRef lets components pass a ref to a child component, which is useful for accessing DOM elements in child components or when building reusable component libraries.

```
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="fancy-button" {...props}>
    {props.children}
  </button>
));

// Parent component can now access the button DOM element
function Parent() {
  const buttonRef = useRef(null);

  useEffect(() => {
    if (buttonRef.current) {
      buttonRef.current.focus();
    }
  }, []);

  return <FancyButton ref={buttonRef}>Click me</FancyButton>;
}
```

Frequency: Medium (55%)

Companies: Companies building component libraries, applications with complex interaction requirements

74. What is the difference between a controlled and uncontrolled form?

Answer:

- Controlled Form: Form element values are controlled by React state. Any input change updates state, and the state value is displayed in the form element.
- Uncontrolled Form: Form elements maintain their own internal state. Values are accessed directly from the DOM using refs.

Controlled forms are more "React-like" and give more control, while uncontrolled forms can be simpler for basic forms.

Frequency: High (85%)
Companies: All companies using React with forms

75. What is React Concurrent Mode?

Answer: Concurrent Mode is a set of features in React 18 that help applications stay responsive by rendering component trees without blocking the main thread. It allows React to:

- Interrupt rendering to handle more urgent updates
- Skip unnecessary renders
- Show fallback UI during long-running operations (Suspense)
- Coordinate transitions with useTransition and useDeferredValue

It's not a separate mode anymore in React 18 but integrated as core features.

Frequency: Medium (55%)
Companies: Facebook, Netflix, apps focused on smooth user experiences