Basic Questions:

1. `Introduction & Basic Questions (15 mins)`

Questions:

1. **Tell me about yourself and what excites you about React?**
    - (This question is to understand your motivation and passion for React and frontend development.)
2. **What is React?**
    - React is a JavaScript library for building user interfaces, specifically for single-page applications where you need a fast, interactive experience. It allows us to create reusable UI components and manage state efficiently.
3. **Can you explain what JSX is in React?**
    - JSX (JavaScript XML) is a syntax extension for JavaScript that looks similar to HTML but allows us to write HTML-like code within JavaScript. React components return JSX to render UI. Behind the scenes, JSX is converted to regular JavaScript calls (e.g., `React.createElement()`).
4. **What are components in React?**
    - Components are the building blocks of a React application. A component is a function or a class that optionally accepts inputs (called "props") and returns a React element that describes how a UI should appear.
5. **Explain the difference between functional components and class components.**
    - **Functional Components**: Simple JavaScript functions that receive props and return JSX. They are easier to write and are the recommended way in

modern React.

- **Class Components**: ES6 classes that extend `React.Component`. They have lifecycle methods and can manage state, but are less common with React's introduction of hooks.

---

2. `Theoretical Questions on React (20 mins)`

Questions:

1. **What is the virtual DOM in React, and how does it work?**
   - The virtual DOM is a lightweight in-memory representation of the actual DOM. React creates a virtual DOM and compares it with the real DOM (a process called "reconciliation"). If there are any changes, React efficiently updates only the changed parts of the real DOM, which optimizes performance.
2. **What are props in React?**
   - Props (short for "properties") are read-only data passed from parent components to child components. They are immutable and allow components to communicate with each other.
3. **What is state in React, and how is it different from props?**
   - State is a mutable object that is local to a component and can change over time. When state changes, the component re-renders. Unlike props, which are passed from parent to child, state is managed within the component itself.
4. **Explain the concept of 'lifting state up' in React.**
   - Lifting state up refers to the process of moving state from a child component to the nearest common ancestor component. This allows

sibling components to communicate via shared state in the parent component.

5. **What are React hooks, and why are they important?**
    - React hooks allow you to use state and other React features (like lifecycle methods) in functional components. `useState`, `useEffect`, `useContext`, and others are some examples. Hooks make functional components more powerful and allow them to have state and lifecycle methods without needing class components.
6. **What is the useEffect hook, and when would you use it?**
    - `useEffect` is a hook that runs side effects in a function component. It can be used for operations like data fetching, subscriptions, or manually updating the DOM. It runs after the render, but you can control when it runs by passing dependencies.
7. **Explain the concept of "keys" in React lists.**
    - Keys are unique identifiers for elements in a list. They help React efficiently identify which items have changed, been added, or removed. Without keys, React would re-render the entire list unnecessarily, leading to performance issues.

---

3. `React Component Design & Problem Solving (30 mins)`

Scenario 1: Component Design

Question: Design a **To-Do List** application in React where a user can:

- Add a task.
- Mark a task as completed.
- Remove a task.

Follow-up:

- How would you manage the state of the list?
- What would the component hierarchy look like?

Expected Answer:

- **Component Structure**: You could have a parent component `ToDoApp` that holds the list of tasks in its state. Inside it, there could be child components like `TaskList` (to render the list) and `Task` (to render individual tasks). You could also have a `TaskInput` component to add new tasks.

```
jsx
Copy code
function ToDoApp() {
  const [tasks, setTasks] = useState([]);

  const addTask = (task) => {
    setTasks([...tasks, { id: Date.now(), text: task, comp
leted: false }]);
  };

  const toggleTask = (id) => {
    setTasks(tasks.map(task => task.id === id ? {...task,
completed: !task.completed} : task));
  };

  const removeTask = (id) => {
    setTasks(tasks.filter(task => task.id !== id));
  };

  return (
    <div>
      <TaskInput onAdd={addTask} />
      <TaskList tasks={tasks} onToggle={toggleTask} onRemo
ve={removeTask} />
    </div>
```

```
  );
}

function TaskInput({ onAdd }) {
  const [task, setTask] = useState('');

  const handleAdd = () => {
    if (task) {
      onAdd(task);
      setTask('');
    }
  };

  return (
    <div>
      <input value={task} onChange={(e) => setTask(e.targe
t.value)} />
      <button onClick={handleAdd}>Add Task</button>
    </div>
  );
}

function TaskList({ tasks, onToggle, onRemove }) {
  return (
    <ul>
      {tasks.map(task => (
        <Task key={task.id} task={task} onToggle={onToggl
e} onRemove={onRemove} />
      ))}
    </ul>
  );
}

function Task({ task, onToggle, onRemove }) {
  return (
    <li>
```

```
      <span
        style={{ textDecoration: task.completed ? 'line-th
rough' : 'none' }}
        onClick={() => onToggle(task.id)}>
        {task.text}
      </span>
      <button onClick={() => onRemove(task.id)}>Remove</bu
tton>
    </li>
  );
}
```