Sheryians Coding School

Live Cohort

React week

1. Higher Order Components (HOC)

🔹 Definition

A Higher Order Component (HOC) is a function that takes a component and returns a new component with added behavior. It's used to reuse component logic.

🔹 Code (Example)

```
function withLoading(Component) {
  return function WithLoadingComponent({ isLoading, ...props }) {
    if (isLoading) return <p>Loading...</p>;
    return <Component {...props} />;
  };
}
```

🔹 Use Case

Used for features like authentication, conditional rendering, and theming across components.

🔹 Interview Q&A

Q: What is an HOC in React?

A: A function that takes a component and returns an enhanced version of that component.

Q: How do you use HOCs to reuse logic?

A: By wrapping components with an HOC, you can inject shared behaviors (e.g., loading, auth checks).

2. Reusing Components, Lists, and Keys

🔹 Definition

React allows reusing components by passing different props. Lists are rendered from arrays, and keys help React optimize rendering.

🔹 Code (Example)

```
function UserCard({ user }) {
  return <div>{user.name}</div>;
}

function UserList({ users }) {
  return (
    <div>
      {users.map(user => <UserCard key={user.id} user={user} />)}
    </div>
  );
}
```

🔹 Use Case

Displaying data lists like users, blog posts, products, etc.

🔷 Interview Q&A

Q: Why are keys important in React lists?

A: Keys help React identify which items changed, added, or removed for efficient re-rendering.

Q: Can we reuse components with different props?

A: Yes, props make components reusable by customizing their behavior.

3. Props Drilling

🔷 Definition

- Props drilling is the process of passing data from a parent to a deeply nested child component through intermediate components.

🔷 Code (Example)

```
<App>
  <Profile user={user} />
</App>

function Profile({ user }) {
  return <ProfileDetails user={user} />;
}
```

🔹 Use Case

Passing data like user info or theme settings deep in the component tree.

🔹 Interview Q&A

Q: What is props drilling?

A: Passing props through multiple layers just to reach a deeply nested component.

Q: How can we avoid props drilling?

A: Use Context API or state management libraries like Redux.

4. Component Lifecycle (Mounting, Updating, Unmounting)

🔹 Definition

Every React class component goes through three phases:

- Mounting: Added to DOM
- Updating: Re-render due to state/props
- Unmounting: Removed from DOM

🔷 Code (Example):

```
class MyComponent extends React.Component {
  componentDidMount() {
    console.log("Mounted");
  }
  componentDidUpdate() {
    console.log("Updated");
  }
  componentWillUnmount() {
    console.log("Unmounted");
  }
  render() {
    return <div>Hello</div>;
  }
}
```

🔷 Use Case

- Use lifecycle methods for network calls, subscriptions, and cleanups.

🔹 Interview Q&A

Q: What are React lifecycle phases?

A: Mounting, Updating, Unmounting.

Q: When is componentDidMount called?

A: After the component is inserted into the DOM.

5. React Lifecycle Methods

Definition

- Specific methods in class components to perform side-effects during different lifecycle phases.

🔹 Code (Example):

- componentDidMount: Called once after mounting — good for API calls.
- componentDidUpdate: Called after each re-render.
- componentWillUnmount: Used for cleanup.

🔹 Use Case

- Timers, subscriptions, API requests, resource cleanup.

🔹 Interview Q&A

Q: Which lifecycle method is used for API calls?

A: componentDidMount.

6. React Hooks

Definition

- Hooks let you use state and lifecycle methods in functional components.

🔹 Code (Example):

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


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

🔹 Use Case

- Write cleaner functional components without class boilerplate. Helps reuse logic via custom hooks.

🔹 Interview Q&A

Q: Why were hooks introduced?

A: To use state/lifecycle in functional components and avoid class complexity.

7. Rules of Hooks

Definition

Hooks must follow certain rules to work properly.

Rules:

Call only at the top level (no loops, conditions).

Call only from React functions (not normal JS).

🔹 Use Case

Ensures predictable hook behavior and avoids bugs.

🔹 Interview Q&A

Q: Why can't hooks be called conditionally?

A: React depends on hook order — calling them conditionally breaks this.

8. Commonly Used Hooks

Definition

Frequently used built-in hooks provided by React.

Code (Example)

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

// useEffect
useEffect(() => fetchData(), []);

// useContext
const value = useContext(MyContext);

// useRef
const inputRef = useRef();
```

```
// useCallback
const memoizedCallback = useCallback(() => doSomething(), [deps]);

// useMemo
const memoizedValue = useMemo(() => computeValue(), [a, b]);
```

Use Case:

Managing state, side effects, context, performance optimization.

Interview Questions & Answers:

Q: What does useRef do?

A: Stores mutable values that persist across renders without causing re-renders.

9. Custom Hooks

Definition

A custom hook is a function that starts with use and allows you to reuse stateful logic between components.

Code (Example)

```
function useCounter(initial = 0) {
  const [count, setCount] = useState(initial);
  const increment = () => setCount(c => c + 1);
  return { count, increment };
}
```

Use Case:

- Form handling, counters, API fetching, toggle logic, etc.

Q: When should you use custom hooks?

A: When you find repeated logic across multiple components.

Q: What's the benefit of custom hooks?

A: Code reuse and cleaner components.

10. Context API

Definition

Context API lets you share data across the component tree without passing props manually at every level.

Code (Example)

```
const UserContext = React.createContext();

function App() {
  return (
    <UserContext.Provider value={{ name: "Alice" }}>
      <Profile />
    </UserContext.Provider>
  );
}
```

```jsx
function Profile() {
  const user = useContext(UserContext);
  return <div>{user.name}</div>;
}
```

Use Case:

Global state like user authentication, theme, language, etc.

Interview Questions & Answers:

Q: What problem does Context API solve?

A: Avoids props drilling for global/shared data.

Q: How do you use useContext?

A: By calling useContext(SomeContext) inside a function component.

11. Redux and State Management

Definition

Redux is a state container that manages global state for JavaScript apps using actions, reducers, and a store.

Code (Example):

```
const INCREMENT = 'INCREMENT';
const increment = () => ({ type: INCREMENT });

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

Use Case:

Complex apps where state is shared between many components — like dashboards or ecommerce.

Interview Questions & Answers:

Q: What are the core principles of Redux?

A: Single source of truth, state is read-only, and changes through pure functions.

Q: What is a reducer in Redux?

A: A pure function that takes current state and action, and returns new state.

Q: Why use Redux Toolkit?

A: Reduces boilerplate and makes Redux easier with utilities like createSlice.