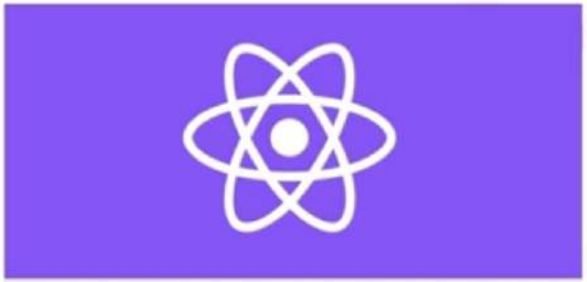
Skills Network

Function Component Lifecycle

IBM

What you will learn

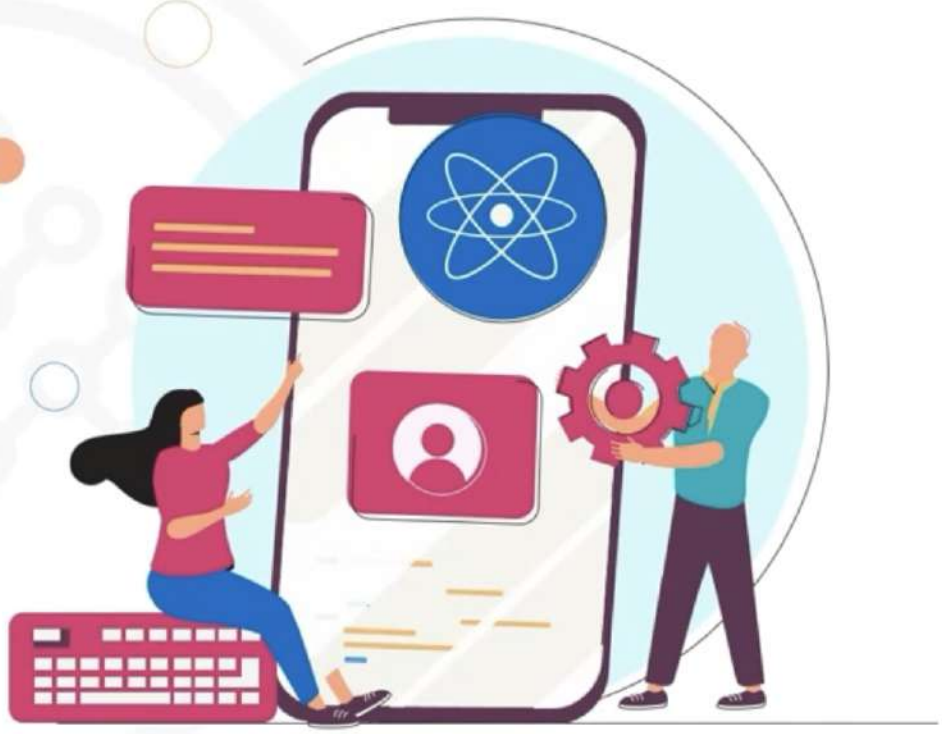


Define functional components in React

Recognize the four different phases of a functional component in React

Functional components in React

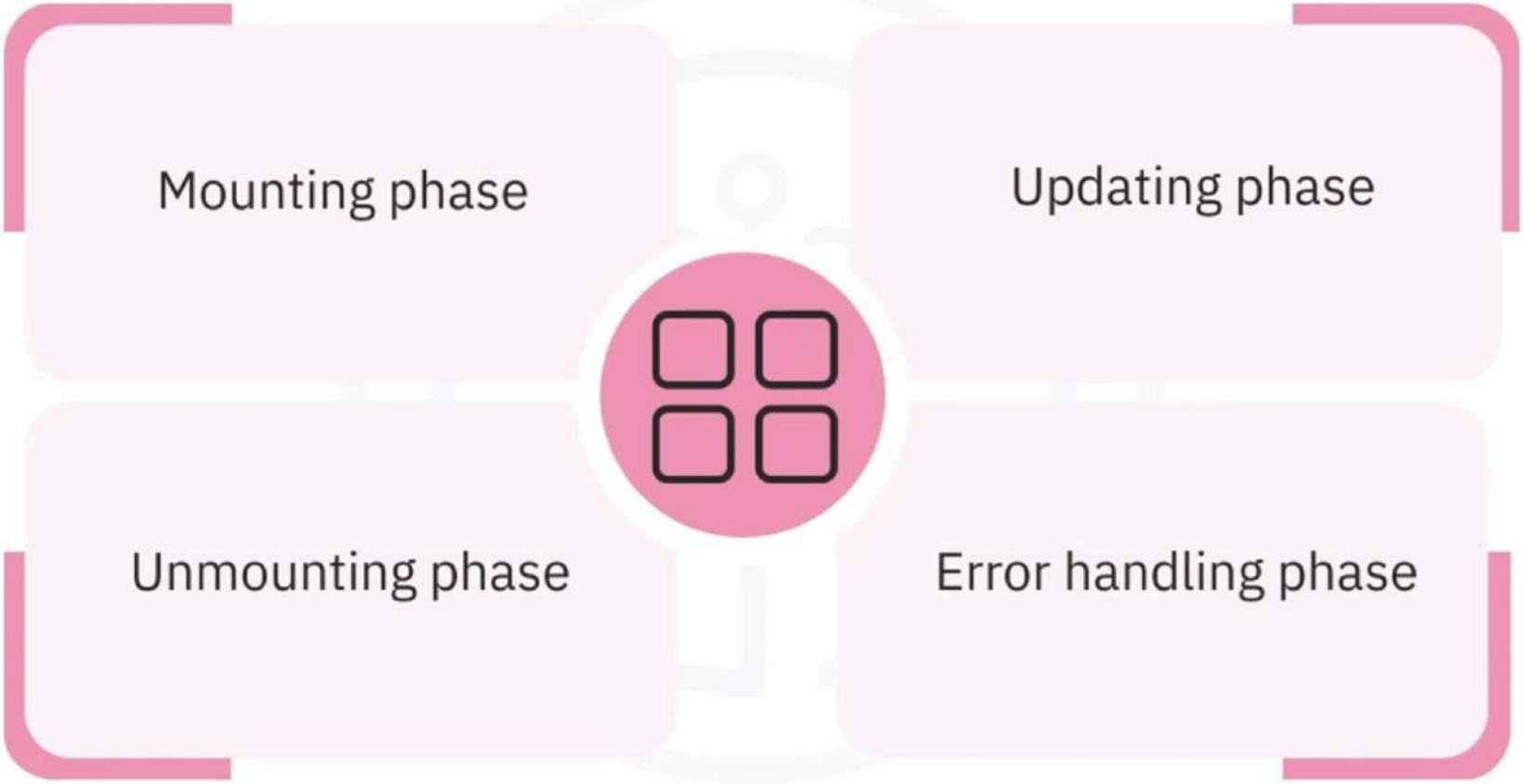
- Functional components are building blocks for UI
- Lifecycle understanding is crucial for:
  - Managing behavior and state of components



Functional components in React

- Functional components lack traditional lifecycle methods
- Similar functionalities achieved with hooks such as:
  - useState, useEffect, and useReducer

Phases of functional component lifecycle



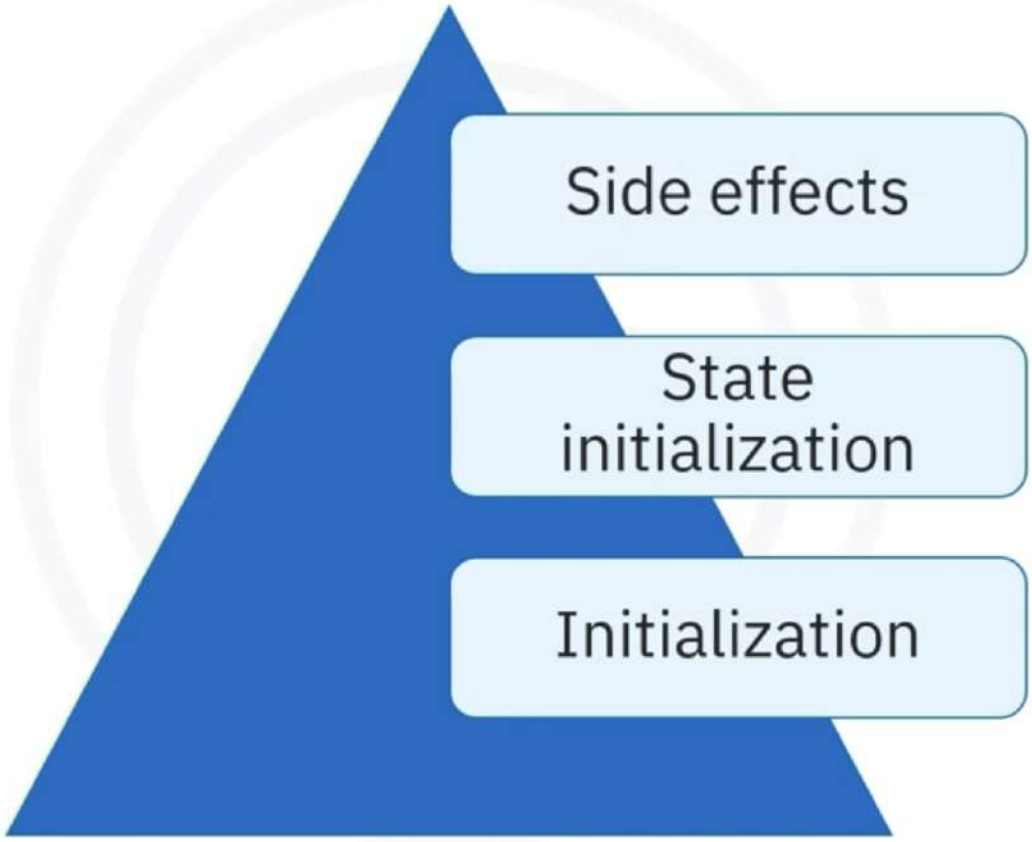
Mounting phase

React in the mounting phase:

- Initializes the functional component
- Prepares it for rendering on the DOM

Mounting phase

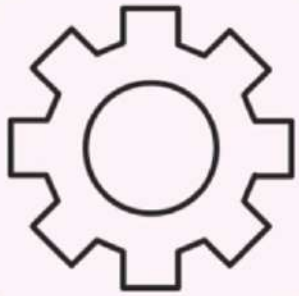
Steps involved:



Steps: Mounting phase

Initialization:

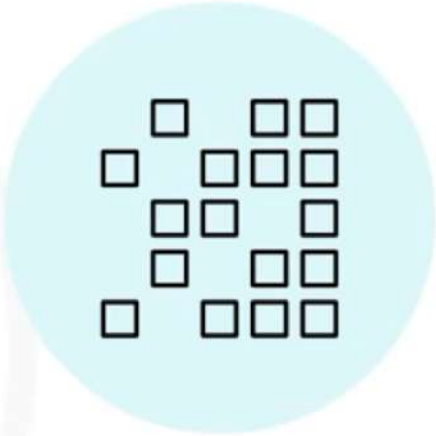
- React:
  - Runs function body of the functional component
  - Sets up the initial structure and behavior



Steps: Mounting phase

State initialization:

- React:
  - Utilizes the useState hook
  - Declares and initializes state variables
- Variables hold data, triggering re-renders



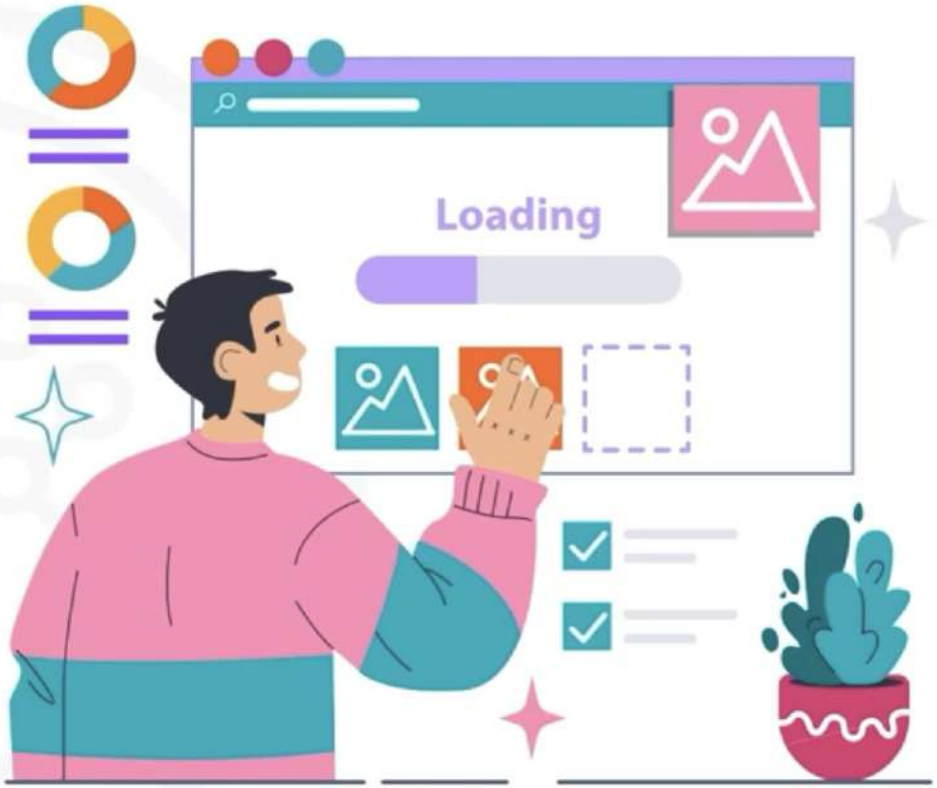
Example: Initialization and state initialization

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

function MyComponent() {
 // State initialization using useState hook
 const [count, setCount] = useState(0);

 return (
  <div>
   <h1>Count: {count}</h1>
  </div>
 );
}
```

Side effects

React:

- Includes data fetching, subscriptions, or DOM manipulation
- Utilizes the useEffect hook with ([])
- Ensures that side effects execute only once
- Optimizes performance and prevents unnecessary re-execution



Side effects: Example

```
useEffect(() => {
  fetch('https://api.example.com/data')
   .then(response => response.json())
   .then(data => {
    setData(data);
   })
   .catch(error => console.error('Error fetching data:',
     error));
}, []);
```

Updating phase

React:

- Responds to changes in the component's state
- Props by re-invoking function body of the component
- Triggers a re-evaluation of JSX

Example: Updating phase

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

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

 const increment = () => {
  setCount(prevCount => prevCount + 1);
 };

 return (
  <div>
   <h1>Count: {count}</h1>
   <button onClick={increment}>Increment</button>
  </div>
 );
}
```

Updating phase

- Responds to user actions and state modifications
- Ensures your UI stays in sync with the underlying data

Unmounting phase

- Involves cleanup operations when removing a component
- Includes cleaning up event listeners, subscriptions, timers

Example: Unmounting phase

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

function MyComponent() {
 useEffect(() => {
  const timer = setInterval(() => {
   console.log('Interval tick');  }, 1000);

  return () => {
   clearInterval(timer); // Cleanup interval on unmount
  };
 }, []);

 return (
  <div>
   <h1>Component with Interval</h1>
  </div>
 );
}
```

Error handling

- Involves routing error to the nearest error boundary
- Is the final phase of the functional component's lifecycle

Error boundaries:

- Catch errors during the rendering phase
- Display a fallback UI
- Ensures application remains functional despite the error

Recap

In this video, you learned that:

- React's lifecycle phases include mounting, updating, unmounting, and error handling
- The mounting phase initializes the component, sets up the initial state, and performs side effects
- In the updating phase, React re-invokes the function body and re-evaluates JSX
- In the unmounting phase, React executes cleanup operations when removing a component from the DOM
- React handles the error by routing it to the nearest error boundary