ReactJs Assigment(2)

Module – 4 Lists and Hooks

Q. Explain Life cycle in Class Component and functional component with Hooks

A. In React, components are the building blocks of your user interface, and they can be classified into two main types: class components and functional components. Both types can have a lifecycle, but they are managed differently. With the introduction of React Hooks in functional components, the way you manage the component lifecycle has become more consistent between the two types.

Class Component Lifecycle:

Class components have a series of lifecycle methods that you can override to control the component's behavior throughout its life. Some of the most commonly used lifecycle methods include:

1. Mounting Phase:

- `constructor(props)`: This is the component's constructor function, where you can initialize state and bind event handlers.
- `componentDidMount()`: This method is called after the component is rendered to the DOM. It's often used for performing initial setup, data fetching, or integrating with third-party libraries.

2. Updating Phase:

- `componentDidUpdate(prevProps, prevState)`: This method is called after a component's props or state change and after re-rendering. You can use it to perform side effects when the component updates.

3. Unmounting Phase:

- `componentWillUnmount()`: This method is called just before the component is removed from the DOM. It's used to clean up resources, event listeners, or subscriptions to prevent memory leaks.

Functional Component with Hooks Lifecycle:

Functional components can mimic the lifecycle behavior of class components using React Hooks. Some of the key hooks that are commonly used to manage the component lifecycle include:

1. Mounting Phase:

   - `useState()`: This hook allows you to add and manage state variables in functional components.

   - `useEffect()`: You can use this hook to perform side effects and mimic the behavior of `componentDidMount`. It runs after rendering and can have a cleanup function.

2. Updating Phase:

   - `useEffect()`: This same hook can be used to mimic `componentDidUpdate` by specifying dependencies in the dependency array. It will run whenever the dependencies change.

3. Unmounting Phase:

   - `useEffect()`: You can mimic the behavior of `componentWillUnmount` by returning a cleanup function from the `useEffect` hook. This function will run when the component unmounts.