Life cycle of react application

The lifecycle of a React application refers to the various stages and events that occur during the lifespan of a React component. React components have different lifecycle methods that you can use to perform actions at specific points in their lifecycle. With the introduction of React Hooks in React 16.8, functional components also have access to these lifecycle events through functions like `useEffect`. Below, I'll outline the key phases and lifecycle methods of a React component:

1. **Mounting Phase**:
   - **constructor**: This is the first method called when a component is created. You can initialize state and bind event handlers here.
   - **render**: The `render` method is responsible for rendering the component's JSX structure. It must be a pure function and should not have side effects.
   - **componentDidMount**: This method is called immediately after a component is inserted into the DOM. It's a good place to initiate network requests, set up event listeners, or perform other setup tasks.
2. **Updating Phase**:
   - **render**: The `render` method is called again when the component's state or props change.
   - **componentDidUpdate**: This method is called after a component's update is reflected in the DOM. It can be used for side effects after an update, such as making additional network requests.
3. **Unmounting Phase**:
   - **componentWillUnmount**: This method is called just before a component is removed from the DOM. It's a good place to clean up resources, such as canceling network requests or removing event listeners.
4. **Error Handling Phase** (introduced in React 16):
   - **componentDidCatch**: This method is used to handle errors that occur during rendering. It's primarily used in

class components to catch errors within the component tree.

5. **React Hooks**:
   - With the introduction of React Hooks, functional components can also manage side effects and state changes throughout their lifecycle. The equivalent functions are:
     - **useState**: Allows functional components to manage local component state.
     - **useEffect**: Performs side effects in functional components. It's similar to `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.
     - **useContext**: Provides access to the context API for functional components.
     - And other hooks like `useRef`, `useReducer`, etc., can be used for specific use cases.

Remember that not all components need to implement all lifecycle methods, and the choice of whether to use class components with lifecycle methods or functional components with hooks depends on your project's requirements. Hooks have become the preferred way to handle component lifecycle in modern React applications due to their simplicity and flexibility. However, class components and lifecycle methods are still supported and widely used in existing codebases.