Lists , Hooks , Localstorage , Api Project

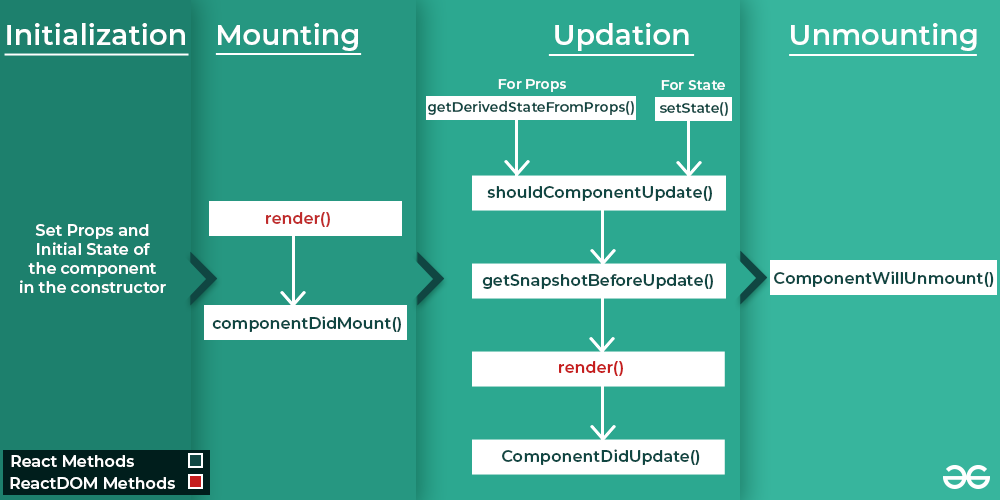
Lifecycle Methods (Class Components)

Question 1: What are lifecycle methods in React class components? Describe the phases of a component’s lifecycle.

Answer 1:

* In React class components, **lifecycle methods** are special functions that let you run code at different stages of a component’s life. A component has three main phases in its lifecycle:

1. Mounting
2. Updation
3. Unmounting



* **Initialization:** (**When the component is created)**
* These methods are called in the Set Props and Initial State of the component in the constructor:
* constructor()
* **Mounting:** **(When the component is added to the DOM)**
* These methods are called in the following order when an instance of a component is being inserted into the DOM:
* [**render()**](https://legacy.reactjs.org/docs/react-component.html#render)
* [componentDidMount()](https://legacy.reactjs.org/docs/react-component.html#componentdidmount)
* **Updating**: **(When the component’s state or props change, causing it to re-render)**
* An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:
* [static getDerivedStateFromProps()](https://legacy.reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [shouldComponentUpdate()](https://legacy.reactjs.org/docs/react-component.html#shouldcomponentupdate)
* [getSnapshotBeforeUpdate()](https://legacy.reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
* [**render()**](https://legacy.reactjs.org/docs/react-component.html#render)
* [componentDidUpdate()](https://legacy.reactjs.org/docs/react-component.html#componentdidupdate)
* **Unmounting:** **(When the component is removed)**
* This method is called when a component is being removed from the DOM:
* [componentWillUnmount()](https://legacy.reactjs.org/docs/react-component.html#componentwillunmount)

Question 2: Explain the purpose of componentDidMount(), componentDidUpdate(),and componentWillUnmount().

Answer 2:

### **Explanation of Lifecycle Methods:**

#### **1. componentDidMount()**

* **When it runs**: After the component is added to the DOM (rendered on the screen).
* **Purpose**: To set up anything your component needs after it’s visible.
* **Uses**:
  + Fetching data from an API.
  + Starting a timer.
  + Subscribing to events.

#### **2. componentDidUpdate()**

* **When it runs**: After the component updates (re-renders) because of changes in its props or state.
* **Purpose**: To perform actions based on updates.
* **Uses**:
  + Making an API call when props/state change.
  + Updating the DOM after changes.
  + Reacting to user interactions.

#### **3. componentWillUnmount()**

* **When it runs**: Right before the component is removed from the DOM.
* **Purpose**: To clean up resources and prevent memory leaks.
* **Uses**:
  + Clearing timers.
  + Unsubscribing from events.
  + Canceling network requests.

These methods are key for managing side effects in class components!

Hooks (useState, useEffect)

Question 1: What are React hooks? How do useState() and useEffect() hooks work in functional components?

Answer 1:

* React **hooks** are tools that let functional components do more. They let you use features like **state** (data that changes) and **lifecycle methods** (actions at different times) without needing class components.
* **Hooks Key Overview:**
* **useState**: For state management.
* **useEffect**: For handling side effects (e.g., data fetching, subscriptions).
* Others: useContext, useReducer, useRef, etc.

### **useState()**

* Adds and manages data (state) in a component.
* The component updates when the data changes.
* Example: Track button clicks or toggle themes.

### **useEffect()**

* Handles tasks outside the component (e.g., fetching data or starting a timer).
* Runs on render or when specified values change.
* Can clean up tasks when the component is removed.

### **In short**

* **useState()**: Manage data.
* **useEffect()**: Handle side tasks.

Question 2: What problems did hooks solve in React development? Why are hooks considered an important addition to React?

Answer 2:

### **Problems Hooks Solved in React Development:**

* **State management in functional components**: Before hooks, only class components could have state. Hooks like useState allow functional components to manage state.
* **Reusability of logic**: Hooks like useEffect let you reuse logic between components without changing their structure.
* **Cleaner code**: They remove the need for lifecycle methods in class components, simplifying code.
* **Easier to understand**: Hooks make functional components more powerful, reducing complexity compared to class components.

### **Why Hooks Are Important**:

* **Simplify Development**: Hooks make functional components as powerful as class components, but with less complexity.
* **Cleaner Code**: They reduce boilerplate code, making components easier to read and maintain.
* **Better Reusability**: Custom hooks allow logic to be reused across components.
* **Improved Side Effect Management**: useEffect() helps keep side effects clean and easy to manage in one place.

In short, hooks make React development simpler, more efficient, and easier to maintain!

Forms in React

Question 1: How do you handle forms in React? Explain the concept of controlled components.

Answer 1:

* To handle forms in React, you typically use **controlled components** where the form elements' values are managed by React state.
* **Steps to Handle Forms:**

1. **Create State**: Use useState to track input values.
2. **Bind Input Value**: Set the form field’s value to the state.
3. **Handle Changes**: Use onChange to update the state when the user types.
4. **Handle Submit**: Use onSubmit to process the form data.

* **Controlled Components**:
* **Controlled components** are form elements like inputs or textareas where React controls the value.
* React stores the value in **state** rather than the input itself.
* This keeps the input’s value and React’s state in sync.
* React manages the form data, making it more predictable.
* You can easily **validate input** or handle changes in real-time.
* React can set **default values** and dynamically change the behavior of the input.
* **Single source of truth**: React’s state is the only place where the input value is stored.
* Controlled components make it easier to manage and update form data.
* They require more code but give you **better control** over form behavior.

Question 2: What is the difference between controlled and uncontrolled components in React?

Answer 2:

**Difference between controlled and uncontrolled components**

**Controlled Components:**

* React controls the input.
* The input's value is stored in React's state.
* Every time you type something, React updates the value.

**Uncontrolled Components:**

* The input controls itself.
* React only checks the value when needed using a reference.
* React doesn’t actively manage the input’s value.

**Main difference:**

* **Controlled:** React manages the input all the time.
* **Uncontrolled:** The input works on its own, and React checks it occasionally.