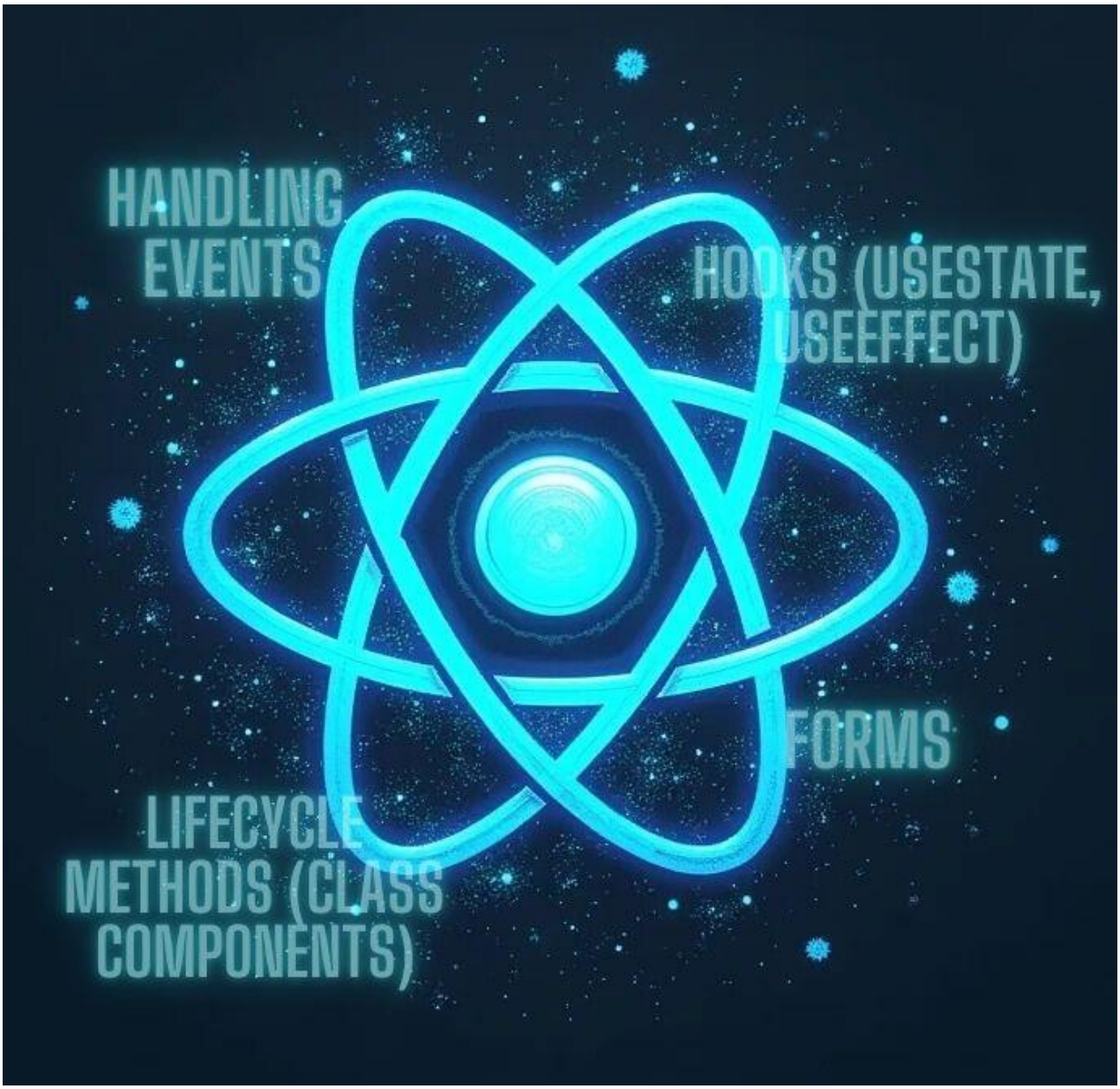

FORMS, HANDLING EVENTS, HOOKS (USESTATE, USEEFFECT), LIFECYCLE METHODS(CLASS COMPONENTS)

Forms in React

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

Ans.

Forms in React are managed by **controlled components**, which ensure that form elements (like `<input>`, `<textarea>`, etc.) have their values controlled by the React state. This approach enables synchronization between the UI and the application state.

Concept of Controlled Components :

1. **Definition**:
   Controlled components are input elements whose value is controlled by React's state. The `value` attribute of the form element is bound to the React state, and any change is handled by an event handler (like onChange).
2. **Advantages**:
   a. Provides complete control over form inputs.
   b. Makes data validation and conditional rendering easier.

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

Ans:

1. **Controlled Component**

   A **controlled component** is one where the form element's value is controlled by React's state. The component relies on the `value` prop and updates the state using an `onChange` event handler.

2. **Uncontrolled Components**

   An **uncontrolled component** relies on the **DOM** itself to manage its value, rather than React state. You access the value using a `ref`.

Handling Events in React

Question 1 : How are events handled in React compared to vanilla JavaScript? Explain The concept of synthetic events.

Ans.

1. **Events in Vanilla JavaScript**
   In vanilla JavaScript, events are handled directly using event listeners like `addEventListener`. The DOM element triggers the event, and the event is handled with the listener function.

2. **Events in React**
   React handles events using **synthetic events**, which are a layer of abstraction over native DOM events. Event handlers in React are written as properties of JSX elements using camelCase (e.g., `onClick`).

Concept of Synthetic Events

Synthetic events in React are objects that wrap around the native browser events. They provide a consistent interface across all

browsers, ensuring compatibility and avoiding cross-browser inconsistencies.

Question 2 : What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

Ans.

common event handlers in React.js:

1. **onClick**: Triggered when an element is clicked.

Example:
```
<button onClick={handleClick}>Click me</button>
```

2. **onChange**: Triggered when the value of an input field changes.

Example:
```
<input type="text" onChange={handleInputChange} />
```

3. **onSubmit**: Triggered when a form is submitted.

Example:
```
<form onSubmit={handleSubmit}>...</form>
```

4. **onMouseOver**: Triggered when the mouse pointer hovers over an element.

5. **onMouseOut**: Triggered when the mouse pointer leaves an element.

6. **onKeyDown**: Triggered when a key is pressed.

7. **onKeyUp**: Triggered when a key is released.

8. **onFocus**: Triggered when an element gains focus.

9. **onBlur**: Triggered when an element loses focus.

10. **onDoubleClick**: Triggered when an element is double-clicked.

Question 3 : Why do you need to bind event handlers in class components ?

Ans.

In React class components, event handlers often need to be **bound** to the component instance because the `this` keyword in JavaScript behaves differently depending on the context of the function call. Without binding, the `this` keyword in event handlers may be `undefined` or reference the wrong context.

Hooks (useState, useEffect)

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

Ans.

React Hooks are functions introduced in React 16.8 that let you use state and other React features in **functional components**, making them powerful and versatile. Before hooks, managing state and lifecycle methods was only possible in class components.

How Does `useState` Work?

The `useState` hook allows you to add state to functional components. It returns :

1. **State Value**: The current state.
2. **State Setter Function**: A function to update the state.

Syntax :

```
const [state, setState] = useState(initialValue);
```

How Does `useEffect` Work?

The `useEffect` hook lets you perform **side effects** in functional components. Side effects include fetching data, subscribing to events, or directly interacting with the DOM.

Syntax :

```
useEffect(() => {
            // Side effect logic here
            return () => {
                // Cleanup logic (optional)
            };
        }, [dependencyArray]);
```

- **Callback Function**: Runs after the component renders.
- **Cleanup Function**: (Optional) Runs when the component unmounts or before the effect re-runs.
- **Dependency Array**: Specifies when the effect should re-run. Leave it empty ([ ]) to run the effect only once (like `componentDidMount`).

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

Ans.

React Hooks, introduced in version 16.8, addressed several issues and limitations of class components and older patterns in React. They simplified the way developers manage state, lifecycle methods, and logic reuse in functional components.

An important addition to the hooks for React is considered

- Simplify state and side-effect management.
- Enable reusable and modular logic.
- Reduce boilerplate code.
- Empower functional components to replace class components.

LIFECYCLE METHODS(CLASS COMPONENTS)

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

Ans.

Lifecycle methods are special functions in React class components that are invoked at different stages of a component's existence. These methods enable developers to execute code at specific points in a component's lifecycle, such as when it's created, updated, or destroyed.

Phases of a React Component's Lifecycle :

1. **Mounting**: When the component is being inserted into the DOM.
2. **Updating**: When the component is being re-rendered due to changes in state or props.
3. **Unmounting**: When the component is being removed from the DOM.
4. **Error Handling**: When an error occurs during rendering or lifecycle methods.

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

Ans.

These lifecycle methods in React class components are essential for handling side effects and managing component behavior during specific stages of its lifecycle.

Method	When It Is Called	Common Use Cases
`componentDidMount()`	After the component is inserted into the DOM	Initializing state, data fetching, DOM manipulation
`componentDidUpdate()`	After props or state changes	Fetching new data, updating DOM based on stateprops
`componentWillUnmount()`	Just before the component is removed	Cleaning up timers, event listeners, or subscriptions