**Controlled Component :**

Controlled and uncontrolled components refer to two different approaches for handling form data in React applications.

Let me explain these concepts:

**Controlled Components**

A controlled component is a component that renders form elements and controls their values through React's state. Here's how it works:

1. **State Management**: The form's data is stored in the component's state.
2. **Event Handling**: Any changes to the form's data are handled by event handlers that update the state.
3. **Rendering**: The form elements' values are set from the component's state, ensuring the form's displayed data is always in sync with the state.

**Example**

jsx

import React, { useState } from 'react';

function ControlledComponent() {

const [value, setValue] = useState('');

const handleChange = (event) => {

setValue(event.target.value);

};

return (

<div>

<input type="text" value={value} onChange={handleChange} />

<p>{value}</p>

</div>

);

}

export default ControlledComponent;

**Uncontrolled Components**

An uncontrolled component is a component that renders form elements but does not control their values through React's state. Instead, the form's data is managed by the DOM directly.

1. **DOM Management**: Form data is stored in the DOM and accessed using references (refs) in React.
2. **Less Event Handling**: Changes to the form's data are managed by the form elements themselves without much intervention from React.
3. **Accessing Data**: To access the form data, you use refs to directly read the DOM elements' values.

**Example**

jsx

import React, { useRef } from 'react';

function UncontrolledComponent() {

const inputRef = useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

alert(`Input value: ${inputRef.current.value}`);

};

return (

<form onSubmit={handleSubmit}>

<input type="text" ref={inputRef} />

<button type="submit">Submit</button>

</form>

);

}

export default UncontrolledComponent;

**Key Differences**

* **State Management**: Controlled components store and manage form data in React state, while uncontrolled components store and manage form data in the DOM.
* **Event Handling**: Controlled components use event handlers to manage form data updates, whereas uncontrolled components rely more on refs and the DOM.
* **Complexity**: Controlled components can offer better data management and validation, but they can be more complex to implement. Uncontrolled components are simpler but less flexible.

**controlled component another example :**

A "controlled component" in React is a form element (like an input field or a select dropdown) where the value is entirely managed by the component's state, meaning the value of the element is always directly tied to a piece of state within the component, and any changes to the input are reflected immediately in the state due to an event handler function; this provides a robust way to handle form data validation and updates in React.

Example:

Code

import React, { useState } from 'react';

function ControlledComponent() {

const [inputValue, setInputValue] = useState('');

const handleChange = (event) => {

setInputValue(event.target.value); // Update state on input change

};

return (

<div>

<input

type="text"

value={inputValue} // Value is directly from state

onChange={handleChange}

/>

<p>Current Value: {inputValue}</p>

</div>

);

}

export default ControlledComponent;

Explanation:

* useState Hook:

The useState hook creates the inputValue state variable which will store the current value of the input field.

* handleChange Function:
  + This function is called whenever the user types in the input field.
  + It gets the new value from the event (event.target.value) and updates the inputValue state using setInputValue.
* value prop:

The key aspect of a controlled component is setting the value prop of the input element to the state variable (inputValue). This ensures that the input field always displays the value stored in the state.

* onChange prop:

The onChange prop is attached to the handleChange function, which allows React to update the state when the user interacts with the input field.

Key points about controlled components:

* **Validation:**

Since the value is always in the state, you can easily add validation logic to the handleChange function before updating the state.

* **Data consistency:**

Controlled components guarantee that the UI always reflects the latest state of the application, preventing inconsistencies.

* **Form submission:**

When submitting a form, you can easily access all form data from the component state.

**Uncontrolled Component :**

An uncontrolled component in React is one where the form data is handled by the DOM rather than the component's state. It relies on refs to access and interact with the form data. Here's an example to illustrate this concept:

**Example of an Uncontrolled Component**

jsx

import React, { useRef } from 'react';

function UncontrolledComponent() {

const inputRef = useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

alert(`Input value: ${inputRef.current.value}`);

};

return (

<form onSubmit={handleSubmit}>

<label>

Name:

<input type="text" ref={inputRef} />

</label>

<button type="submit">Submit</button>

</form>

);

}

export default UncontrolledComponent;

**How It Works**

1. **Reference (ref)**: A useRef hook is used to create a reference to the input element.
2. **Accessing the DOM**: The ref is passed to the input element, allowing us to access its value directly from the DOM.
3. **Event Handling**: When the form is submitted, the handleSubmit function is triggered. It prevents the default form submission behavior and displays an alert with the input value obtained from the ref.

**Key Points**

* **DOM Management**: Form data is handled by the DOM rather than the component's state.
* **Simplicity**: Uncontrolled components can be simpler to implement since they don't require state management.
* **Less Flexibility**: They are less flexible compared to controlled components when it comes to validation and managing form data.

Uncontrolled components are often useful for simple forms or when you need to integrate with third-party libraries that rely on the DOM.

Uncontrolled component –another example :

An uncontrolled component in React is a form element that manages its own state directly through the DOM, rather than being controlled by React's state mechanism; to access its value, you typically use a ref to reference the DOM node and retrieve the value from it when needed, like in a form submission handler.

Example:

Code

import React, { useRef } from 'react';

function UncontrolledForm() {

const inputRef = useRef(null); // Create a ref to access the input element

const handleSubmit = (event) => {

event.preventDefault();

console.log('Input value:', inputRef.current.value); // Access value directly from DOM

};

return (

<form onSubmit={handleSubmit}>

<input type="text" ref={inputRef} />

<button type="submit">Submit</button>

</form>

);

}

export default UncontrolledForm;

Explanation:

* useRef hook:

The inputRef created using useRef is a way to store a reference to the input element on the page.

* **Accessing the value:**

When the form is submitted, the handleSubmit function gets the value of the input field using inputRef.current.value.

Key points about uncontrolled components:

* **No state management:**

Unlike controlled components, you don't need to manage the input value in React state.

* **Simpler setup:**

For simple forms where you only need to access the value on submission, uncontrolled components can be easier to implement.

* **Less validation:**

Since React doesn't directly control the value, validating the input might require more work.

* **Use cases:**
  + File input elements (<input type="file">) are inherently uncontrolled as the browser manages the file selection.
  + When you need a basic input field without complex validation or real-time updates.

let's dive into another example of an uncontrolled component in React. This time, let's create a form with multiple input fields and demonstrate how to handle their values using refs.

**Example of Uncontrolled Component with Multiple Inputs**

jsx

import React, { useRef } from 'react';

function MultiInputUncontrolledComponent() {

const nameRef = useRef(null);

const emailRef = useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

const name = nameRef.current.value;

const email = emailRef.current.value;

alert(`Name: ${name}, Email: ${email}`);

};

return (

<form onSubmit={handleSubmit}>

<div>

<label>

Name:

<input type="text" ref={nameRef} />

</label>

</div>

<div>

<label>

Email:

<input type="email" ref={emailRef} />

</label>

</div>

<button type="submit">Submit</button>

</form>

);

}

export default MultiInputUncontrolledComponent;

**How It Works**

1. **References (refs)**: Two useRef hooks are used to create references for the name and email input elements.
2. **Accessing the DOM**: The refs are passed to the respective input elements, allowing us to access their values directly from the DOM.
3. **Event Handling**: When the form is submitted, the handleSubmit function is triggered. It prevents the default form submission behavior and displays an alert with the values of the name and email inputs obtained from the refs.

**Key Points**

* **DOM Management**: Each form field's data is handled by the DOM rather than the component's state.
* **Simplicity**: Uncontrolled components can be simpler to implement since they don't require state management.
* **Less Flexibility**: They are less flexible compared to controlled components when it comes to validation and managing form data.