Class-05: React Forms and Validation

1. Introduction

Forms are an essential part of web applications, enabling user input and interaction. React provides two ways to handle form inputs: **Controlled Components** and **Uncontrolled Components**. This session covers handling form inputs and implementing basic validation using built-in methods.

2. Controlled and Uncontrolled Components

2.1 Controlled Components

- In **Controlled Components**, React controls the form elements through state.
- The input values are stored in the component's state and updated via event handlers.
- This approach ensures a single source of truth and makes validation easier.

Example:

```
import React, { useState } from 'react';

function ControlledForm() {
  const [inputValue, setInputValue] = useState("");

  const handleChange = (event) => {
    setInputValue(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert("Submitted Value: " + inputValue);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={inputValue} onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
}

export default ControlledForm;
```

2.2 Uncontrolled Components

- In **Uncontrolled Components**, React does not control the input value directly.
- The form data is accessed via `ref` instead of state.
- Useful when integrating React with non-React code or when performance is a concern.

Example:

```
import React, { useRef } from 'react';

function UncontrolledForm() {
  const inputRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    alert("Submitted Value: " + inputRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" ref={inputRef} />
      <button type="submit">Submit</button>
    </form>
  );
}

export default UncontrolledForm;
```

3. Handling Form Inputs

- **onChange Event**: Used to update state when the input value changes.
- **onSubmit Event**: Prevents default form submission and processes data.
- **Multiple Inputs Handling**: Use state objects for multiple form fields.

Example:

```
import React, { useState } from 'react';

function MultiInputForm() {
  const [formData, setFormData] = useState({ name: "", email: "" });

  const handleChange = (event) => {
    setFormData({ ...formData, [event.target.name]: event.target.value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Name: ${formData.name}, Email: ${formData.email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" name="name" value={formData.name}
onChange={handleChange} placeholder="Name" />
      <input type="email" name="email" value={formData.email}
onChange={handleChange} placeholder="Email" />
      <button type="submit">Submit</button>
    </form>
  );
}

export default MultiInputForm;
```

4. Basic Validation (Using Built-in Methods)

4.1 Required Field Validation

- The `required` attribute in HTML ensures that fields are not left empty.

Example:

```
<input type="text" required />
```

4.2 Pattern Validation

- The `pattern` attribute restricts input formats (e.g., email, phone numbers).

Example:

```
<input type="email" pattern=".+@.+\..+" required />
```

4.3 JavaScript-based Validation

- Implemented using state and event handlers.

Example:

```
import React, { useState } from 'react';

function ValidationForm() {
  const [email, setEmail] = useState("");
  const [error, setError] = useState("");
  const handleChange = (event) => {
    setEmail(event.target.value);
    if (!event.target.value.includes("@")) {
      setError("Invalid email format");
    } else {
      setError("");
    }
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    if (!error) {
      alert("Form submitted successfully!");
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="email" value={email} onChange={handleChange}
placeholder="Enter email" />
      {error && <p style={{ color: "red" }}>{error}</p>}
      <button type="submit" disabled={!!error}>Submit</button>
    </form>
  );
}

export default ValidationForm;
```

5. Conclusion

- **Controlled Components** manage form data via state, ensuring React has full control.
- **Uncontrolled Components** use refs for direct DOM manipulation.
- **Handling form inputs** involves `onChange` and `onSubmit` events.
- **Basic validation** can be done using built-in HTML attributes or JavaScript-based methods.

By understanding these concepts, you can effectively handle forms in React applications and enhance user experience with proper validation techniques.