Day 10: Forms & Controlled Components – Inputs, Text Fields, Form Handling

---

1. What are Forms in React?

Forms are essential in web applications because they allow users to interact with the system by submitting data (e.g., login forms, registration forms, search fields). In plain HTML, forms manage their own state (the values entered in the input fields). However, in React, we often want to take full control of the form data to validate, process, and send it to a backend.

---

2. What are Controlled Components?

A **Controlled Component** is a form element (like an input, textarea, or select) whose value is controlled by React state. Instead of the DOM handling the form input values, React takes control. This allows us to: - Synchronize form data with state. - Validate inputs as the user types. - Reset values programmatically. - Dynamically update UI based on input.

Key Idea:

> Controlled components always have their values set from React's state, and any change updates the state.

---

3. Why Do We Use Controlled Components?

- To have **full control** over form inputs.
- To **validate input values** in real time.
- To **conditionally enable/disable buttons** (e.g., disable submit until all fields are valid).
- To **manage multiple inputs** with ease using state.

---

4. Basic Example of Controlled Input

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

function SimpleForm() {
  const [name, setName] = useState("");

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

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Form submitted! Name: ${name}`);
  };

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

export default SimpleForm;
```

Explanation: - `value={name}` → input value comes from React state. - `onChange={handleChange}` → updates state whenever user types. - Controlled input ensures React always knows the current value.

---

5. Handling Multiple Inputs

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

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

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

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
```

```
        Username:
        <input type="text" name="username" value={formData.username}
onChange={handleChange} />
      </label>
      <br />
      <label>
        Email:
        <input type="email" name="email" value={formData.email}
onChange={handleChange} />
      </label>
      <br />
      <button type="submit">Submit</button>
    </form>
  );
}

export default MultiInputForm;
```

Explanation: - `formData` object manages multiple fields. - `name` attribute of input is used as key to update state dynamically.

---

6. Textarea & Select with Controlled Components

```
function SelectTextareaForm() {
  const [bio, setBio] = useState("");
  const [role, setRole] = useState("user");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Bio: ${bio}, Role: ${role}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Bio:
        <textarea value={bio} onChange={(e) => setBio(e.target.value)} />
      </label>
      <br />
      <label>
        Role:
        <select value={role} onChange={(e) => setRole(e.target.value)}>
          <option value="user">User</option>
          <option value="admin">Admin</option>
          <option value="guest">Guest</option>
        </select>
      </label>
      <br />
      <button type="submit">Save</button>
    </form>
  );
}
```

---

7. Form Validation with Controlled Components

```
function ValidatedForm() {
  const [email, setEmail] = useState("");
  const [error, setError] = useState("");

  const handleChange = (e) => {
    const value = e.target.value;
    setEmail(value);
    setError(value.includes("@") ? "" : "Invalid email format");
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!error && email) alert(`Submitted: ${email}`);
  };

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

Explanation: - Validation is performed while typing. - The submit button is disabled if the input is invalid.

📝Exercise (15–20 Minutes)

Goal: Build a complete **Registration Form** using controlled components.

Requirements: 1. Create a form with the following fields: - Full Name (text input) - Email (email input) - Password (password input) - Gender (radio buttons: Male/Female/Other) - Bio (textarea) - Role (select dropdown: User/Admin/Guest) 2. Each input must: - Be **controlled** by React state. - Update state on change. - Show the live value below the form in JSON format. 3. Add simple validation: - Full Name should not be empty. - Email must contain `@`. - Password must be at least 6 characters. - Disable Submit until all fields are valid. 4. On Submit: - Prevent default behavior. - Log all data to console. - Show an alert with success message.

✅By completing this, you'll understand **Forms & Controlled Components** thoroughly.