📘Day 20: Forms with Validation (Basic Validation Logic)

Concept Overview

Forms are essential for collecting user input in web applications. **Validation** ensures that the data entered is correct, complete, and in the proper format before submission.

🔗Why Use Form Validation?

1. **Data Accuracy:** Prevents users from entering invalid or incomplete data.
2. **Security:** Avoids injection attacks and malicious input.
3. **Better UX:** Gives users instant feedback when they make a mistake.
4. **Data Consistency:** Ensures all inputs follow the same pattern or rules.

⚙️Types of Validation

1. **Required Field Validation** – Ensures no field is left empty.
2. **Length Validation** – Checks if the input meets a minimum or maximum length.
3. **Pattern Validation (Regex)** – Ensures the input matches a specific format (like email or password).
4. **Custom Validation** – Logic built manually for unique conditions (e.g., password match check).

🪫 Example: Basic Form Validation in React

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

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

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

  const validate = () => {
    let newErrors = {};
    if (!formData.name) newErrors.name = "Name is required";
    if (!formData.email) {
      newErrors.email = "Email is required";
    } else if (!/^[^@\s]+@[^@\s]+\.[^@\s]+$/.test(formData.email)) {
      newErrors.email = "Invalid email format";
    }
    return newErrors;
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    const validationErrors = validate();
    if (Object.keys(validationErrors).length > 0) {
      setErrors(validationErrors);
    } else {
      setErrors({});
      alert("Form submitted successfully!");
    }
  };

  return (
    <div style={{ padding: 20 }}>
      <h2>Basic Form Validation 🧾</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label>Name:</label>
          <input
            type="text"
            name="name"
            value={formData.name}
            onChange={handleChange}
          />
          {errors.name && <p style={{ color: "red" }}>{errors.name}</p>}
        </div>

        <div>
          <label>Email:</label>
          <input
            type="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
          />
          {errors.email && <p style={{ color: "red" }}>{errors.email}</p>}
        </div>

        <button type="submit">Submit</button>
      </form>
    </div>
  );
}

export default App;
```

📋Step-by-Step Breakdown

1. **State Management:** Store input values and errors in `useState`.

2. **Input Handling:** Update input fields dynamically using `onChange`.
3. **Validation Function:** Define rules for each field (like regex for emails).
4. **Error Display:** Show error messages dynamically when validation fails.
5. **Form Submission:** Prevent submission if there are validation errors.

🧰Advantages of Validation in React

- Keeps UI responsive and interactive.
- Prevents data corruption before sending it to backend.
- Improves trust by preventing unexpected errors.

Exercise: Build Your Own Validated Form

Goal: Create a registration form with the following fields: - Name (required) - Email (required, must be valid format) - Password (required, min 6 chars) - Confirm Password (must match password)

Requirements: - Display inline error messages for each invalid field. - On successful validation, show a success alert or message.

💡 **Hint:** Use the same validation logic pattern from the example above. Try adding regex and length checks for extra challenge.

🗜️Next Step Preview: Day 21 - Advanced Form Handling (Formik / Yup)