Day 12: React Forms & Form Validation

Today, you'll learn how to handle **forms in React**, including **controlled components, form validation, and error handling**. Forms are essential for user input, such as login, registration, and feedback submission.

1️⃣ Controlled vs. Uncontrolled Components

- **Controlled Components** ✅
    - React controls form input values using `useState`.

    - Updates instantly as the user types.

    - Recommended for most React apps.
- **Uncontrolled Components** ❌
    - Uses **HTML's native behavior** (like `document.getElementById`).

    - Does not use React state.

    - Less common in modern React apps.

Example: Controlled Input

```jsx
import { useState } from "react";

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

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <p>Typed: {name}</p>
    </div>
  );
}

export default ControlledForm;
```

✅ **Uses `useState` to manage the input**
✅ **Updates instantly as the user types**

---

2 Handling Form Submission

Example: Basic Form Submission

```jsx
import { useState } from "react";

function SimpleForm() {
  const [email, setEmail] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault(); // Prevent page reload
    alert(`Submitted Email: ${email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        placeholder="Enter email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <button type="submit">Submit</button>
    </form>
  );
}

export default SimpleForm;
```

✅ **Prevents page refresh using `e.preventDefault()`**
✅ **Alerts the submitted email**

---

3 Handling Multiple Inputs

Instead of managing each input separately, use **one state object**.

```jsx
import { useState } from "react";

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

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

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Username: ${formData.username}, Email: ${formData.email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
```

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

export default MultiInputForm;
```

✅ **Handles multiple inputs with a single state**
✅ **Uses name attributes to update values dynamically**

4️⃣ Form Validation (Basic)

- **Check if fields are empty**

- **Show error messages**

```jsx
import { useState } from "react";

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

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

  const validate = () => {
    let errors = {};
    if (!formData.email.includes("@")) errors.email = "Invalid email!";
    if (formData.password.length < 6)
      errors.password = "Password must be at least 6 characters!";
    return errors;
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const validationErrors = validate();
    if (Object.keys(validationErrors).length > 0) {
      setErrors(validationErrors);
    } else {
      alert("Form Submitted Successfully!");
    }
```

```jsx
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        name="email"
        placeholder="Email"
        value={formData.email}
        onChange={handleChange}
      />
      {errors.email && <p>{errors.email}</p>}

      <input
        type="password"
        name="password"
        placeholder="Password"
        value={formData.password}
        onChange={handleChange}
      />
      {errors.password && <p>{errors.password}</p>}

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

export default FormValidation;
```

✅ **Validates email and password**
✅ **Displays error messages**

5️⃣ Form Validation with `react-hook-form`

`react-hook-form` is a library that simplifies form validation.

Installation
```
npm install react-hook-form
```

Example: Using react-hook-form
```jsx
import { useForm } from "react-hook-form";

function HookForm() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  const onSubmit = (data) => alert(JSON.stringify(data));

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

```
      <input {...register("email", { required: "Email is required!" })} />
      {errors.email && <p>{errors.email.message}</p>}

      <input
        {...register("password", {
          required: "Password is required!",
          minLength: { value: 6, message: "Min 6 characters!" },
        })}
      />
      {errors.password && <p>{errors.password.message}</p>}

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

export default HookForm;
```

✅ **Uses register to handle inputs**
✅ **Simplifies validation with built-in rules**

---

6 Summary of Day 12

✔ **Controlled components** use useState
✔ **onChange updates input values dynamically**
✔ **Form validation ensures correct user input**
✔ **react-hook-form simplifies form handling**
✔ **Error messages help guide users**

🚀 **Next Step: Day 13 - React Router (Navigation & Routing)**