React — Form Handling, useState, onChange, Validation

Creating a React App in the Current Folder

If you want to create a React app inside the current folder, add a dot:

```
npx create-react-app .
```

Without the dot, a new folder will be created for the project.

Designing a Signup Page in React

When building a form such as a Signup page, clicking the Submit button refreshes the browser page. This default behavior causes React components to reload.

Problem: Page Refresh on Submit

React apps should not refresh when submitting a form.

Solution: Use `e.preventDefault()`

```
 const handleFormSub = (e) => {
   e.preventDefault();
   console.log("Form Submitted");
};

return (
  <>
    <form onSubmit={handleFormSub}>
      {/* Form fields */}
    </form>
  </>
);
```

Where Code Is Written in React

- Pure JavaScript is written before the `return`.
- Inside `return`, we write JSX (a syntax similar to HTML).

---

useState and Input Lock Issue

If you write an input like:

```
<input value={email} />
```

The input becomes locked because React controls the value fully. To allow typing, React requires an update mechanism.

Solving the Locked Input Issue

Use `onChange` with `setEmail` to update the state as the user types.

```
onChange={(e) => setEmail(e.target.value)}
```

How `e.target.value` Works in React

1. What is `e`?

`e` stands for **event object**.
Whenever a user types in an input field, clicks a button, or interacts with the form, React sends an event object to your function.

Example:

```
onChange={(e) => console.log(e)}
```

This prints the whole event object.

---

2. What is `e.target`?

`e.target` means **the element that triggered the event**.

If you typed in the email input, then:

- `e.target` = the email `<input />` field If you typed in the password input:
- `e.target` = the password `<input />`

So React tells you **which element** is being changed.

---

3. What is `e.target.value`?

`e.target.value` means: **The current value inside that input field.**

Example:

```
<input
  type="text"
  onChange={(e) => console.log(e.target.value)}
/>
```

If the user types:

```
a
ab
abc
```

The output will be:

```
a
ab
abc
```

React gives you **the updated value on every keystroke**.

---

4. Why We Use It With useState

We use it to update React state like this:

```
onChange={(e) => setEmail(e.target.value)}
```

Meaning:

- Read the user's typed value
- Send it to the `setEmail` function
- Update the `email` variable in real time

So the input stays responsive.

Complete Example

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

const SignUp = () => {
  const [email, setEmail] = useState("abc@gmail.com");
  const [password, setPassword] = useState("");

  const handleFormSub = (e) => {
    e.preventDefault();
    console.log("Form Submitted");
  };

  return (
    <>
      <form onSubmit={handleFormSub}>

        <div className="emailSec">
          <label htmlFor="email">Email:</label>
          <input
            type="email"
            name="email"
            value={email}
            placeholder="Enter your email"
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </div>

        <div className="passwordSec">
          <label htmlFor="password">Password:</label>
          <input
            type="password"
            name="password"
            value={password}
            placeholder="Enter your password"
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>

      </form>
    </>
  );
};

export default SignUp;
```

Why Do We Need `onChange`?

React captures input character by character. Each keystroke updates the state.

Example when typing "abc@gmail.com":

- a
- ab
- abc
- abc@
- abc@g
- ...

React continuously updates the value using state.

In vanilla JavaScript, this is not how updates work:

```
const email = document.getElementById("email");
email.innerHTML = "<p>abc@gmail.com</p>";
```

React does not update the DOM directly. Instead, it updates the state.

Tracking User Input for Validation

If we want to ensure the user must enter an email before submitting:

```
<button type="submit" disabled={!email}></button>
```

If `email` is empty, the button remains disabled. This works even without the HTML `required` attribute.

Multiple Conditions for Disabling the Button

The button should be disabled if:

1. Email is empty
2. Password is empty
3. Form is loading

Example:

```
<button type="submit" disabled={!email || !password || isLoading}>
  Sign Up
</button>
```

Loading State Handling

Initially:

```
const [isLoading, setIsLoading] = useState(false);
```

When the form is submitted:

```
const handleFormSub = (e) => {
  e.preventDefault();
  setIsLoading(true);
};
```

This prevents multiple clicks on the submit button.

Summary

- Use `e.preventDefault()` to stop page refresh.
- Use `useState` to store input values.
- Inputs become locked if using `value` without `onChange`.
- Use `onChange` to update state.
- Use state values to enable or disable the Submit button.
- Add loading logic to avoid multiple submissions.