

Section 17: Working with Forms & User Input

Saturday, October 5, 2024 1:25 PM

The screenshot shows a dark-themed React application titled "React Forms". At the top, there's a circular icon with a phone-like symbol. Below it, the title "React Forms" is displayed. The main area contains a "Welcome on board!" message and a note: "We just need a little bit of data from you to get you started! ✨". There are several input fields: "EMAIL" with the value "test@example.com", "PASSWORD" and "CONFIRM PASSWORD" both containing "*****", "FIRST NAME" with the value "Maximilian", and "LAST NAME" with the value "Schwarzmüller". A dropdown menu for "WHAT BEST DESCRIBES YOUR ROLE?" shows "Teacher" selected. A checkbox group for "How did you find us?" includes "GOOGLE" (unchecked), "REFERRED BY FRIEND" (checked), and "OTHER" (unchecked). A small checkbox at the bottom left says "I AGREE TO THE TERMS AND CONDITIONS". At the bottom right are "Reset" and "Sign up" buttons.

Starting Project
ACADE MIND

Working with Forms & User Input

It's Trickier Than It Might Seem

- ▶ What's **Difficult** About Forms?
- ▶ Handling **Form Submission & Validating** User Input
- ▶ Using **Built-in** Form Features
- ▶ Building **Custom** Solutions



Form Submission

Handling submission is relatively **easy**

Entered values can be managed via **state**

Alternatively, they can be extracted via **refs**

Or via **FormData** and native browser features



Input Validation

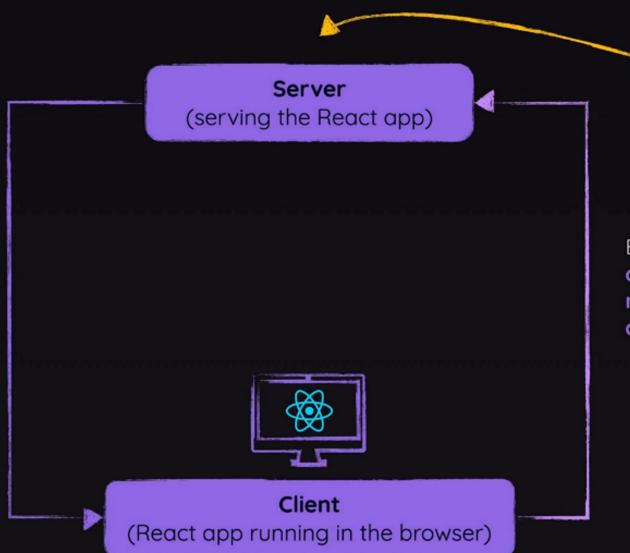
Providing a good user experience is **tricky**

You can **validate** on every **keystroke** → errors may be shown **too early**

You can **validate** on **lost focus** → errors may be shown **too long**

You can **validate** on **form submission** → errors may be shown **too late**

Serves the React app
JavaScript files + index.html
(+ any CSS needed) to users visiting the website



During development, that's the development server started via “`npm run dev`”

Browser **automatically creates & sends a HTTP request** with entered **form data**

The screenshot shows a browser developer tools Network tab. An orange arrow points from the 'Login' form in the main window to the first item in the Network list, which is a request to '?email=&password'. The Network table has columns for Name, S.., Type, I.., S..., T, and Waterfall.

| Name | S.. | Type | I.. | S... | T | Waterfall |
|------------------|------|-------|-----|-------|-----|-----------|
| ?email=&pas... | 3... | do... | O.. | 1... | 5.. | |
| client | 3... | sc... | 2.. | 1... | 4.. | |
| main.jsx | 3... | sc... | 2.. | 1... | 3.. | |
| @react-refresh | 3... | sc... | 2.. | 1... | 2.. | |
| react_jsx-de... | 2... | sc... | m. | (...) | 1.. | |
| react.js?v=f0... | 2... | sc... | m. | (...) | 1.. | |
| react-dom_c... | 2... | sc... | m. | (...) | 5.. | |
| App.jsx | 3... | sc... | m. | 1... | 3.. | |
| index.css | 3... | sc... | m. | 1... | 4.. | |

By default, button type is submit, it sends a req to the server and our page reloads. That's why we do not see the console.log() output to the console.

The screenshot shows a code editor with a file named 'Login.jsx'. A red arrow points from the bottom of the slide towards the 'handleSubmit' function. The code is as follows:

```

src > components > Login.jsx > Login > handleSubmit
1  export default function Login() {
2
3    const handleSubmit = (event) => {
4      event.preventDefault();
5
6      console.log('Submitted');
7
8      return (
9        <form>
10         <h2>Login</h2>
11
12         <div className="control-row">          (property) JSX.IntrinsicElements.label:
13           <div className="control no-margin">          React.DetailedHTMLProps<React.LabelHTMLAttributes<HTMLLabelElement>, HTMLLabelElement>
14             <label htmlFor="email">Email</label>
15             <input id="email" type="email" name="email" />
16           </div>
17
18           <div className="control no-margin">
19             <label htmlFor="password">Password</label>
20             <input id="password" type="password" name="password" />
21           </div>
22         </div>
23
24         <p className="form-actions">
25           <button className="button button-flat">Reset</button>
26           <button className="button" onClick={handleSubmit}>Login</button>
27         </p>
28       </form>
29     );
  
```

Default type is submit, it sends req. to the server.
`<button type="submit" />`

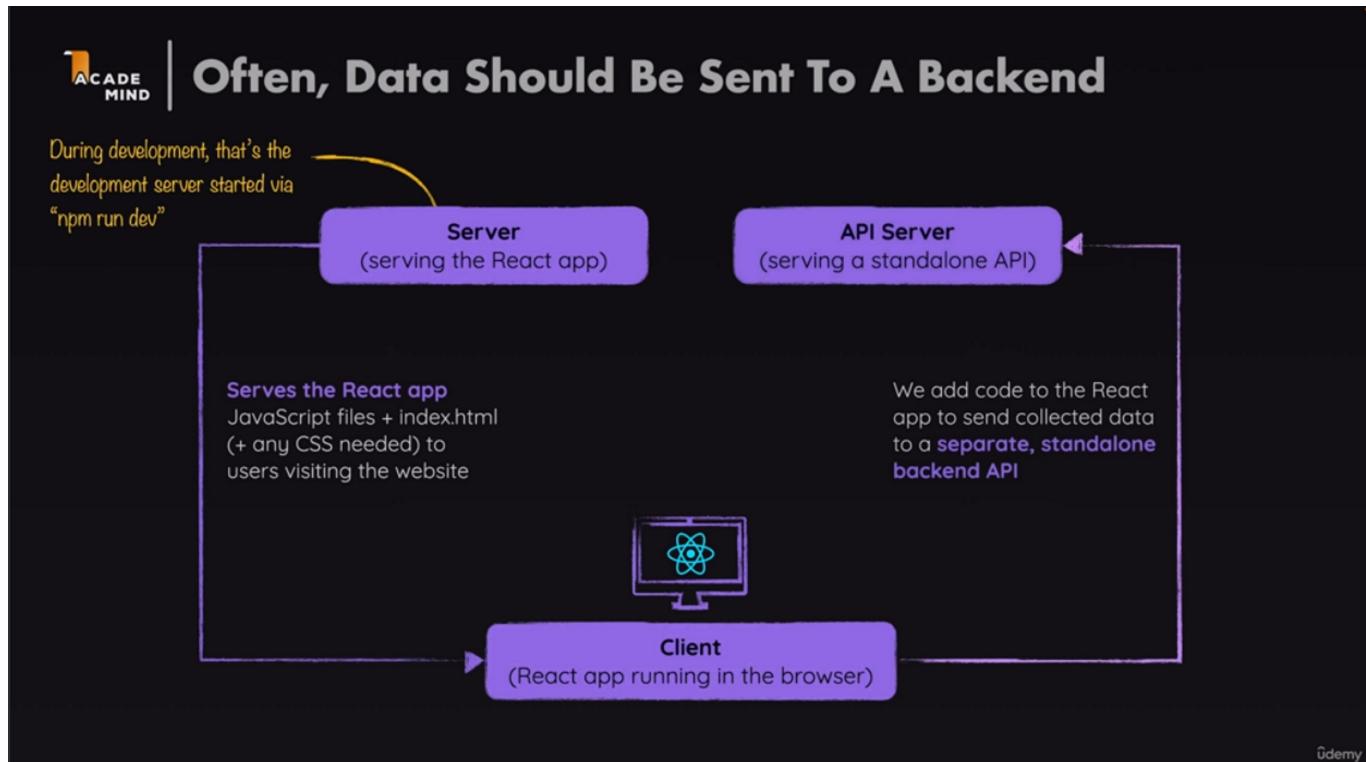
One way of preventing to send req is to add type button

```
<button type="button" className="button" onClick={handleSubmit}>  
| Login  
| </button>  
</p>
```

```
<button type="button" />
```

Another way is adding,

```
event.preventDefault();
```



Resetting Forms

`type='reset'` actually resets our form.

```
1   <label htmlFor="terms-and-conditions">
2     <input type="checkbox" id="terms-and-conditions" name="terms" />I
3     agree to the terms and conditions
4   </label>
5 </div>

6
7
8   <p className="form-actions">
9     <button type="reset" className="button button-flat">
10       Reset
11     </button>
12     <button type="submit" className="button">
13       Sign up
14     </button>
15   </p>
16 </form>
17 :
```

Or we can write like this way,

```
  Signup.jsx M X  StateLogin.jsx  Login.jsx
1  export default function Signup() {
2    function handleSubmit(event) {
3      event.preventDefault();
4
5      const fd = new FormData(event.target);
6      const acquisitionChannel = fd.getAll('acquisition')
7      const data = Object.fromEntries(fd.entries());
8      data.acquisition = acquisitionChannel;
9      console.log(data);
10
11      event.target.reset(); // Red arrow points here
12    }
13
14    return (

```

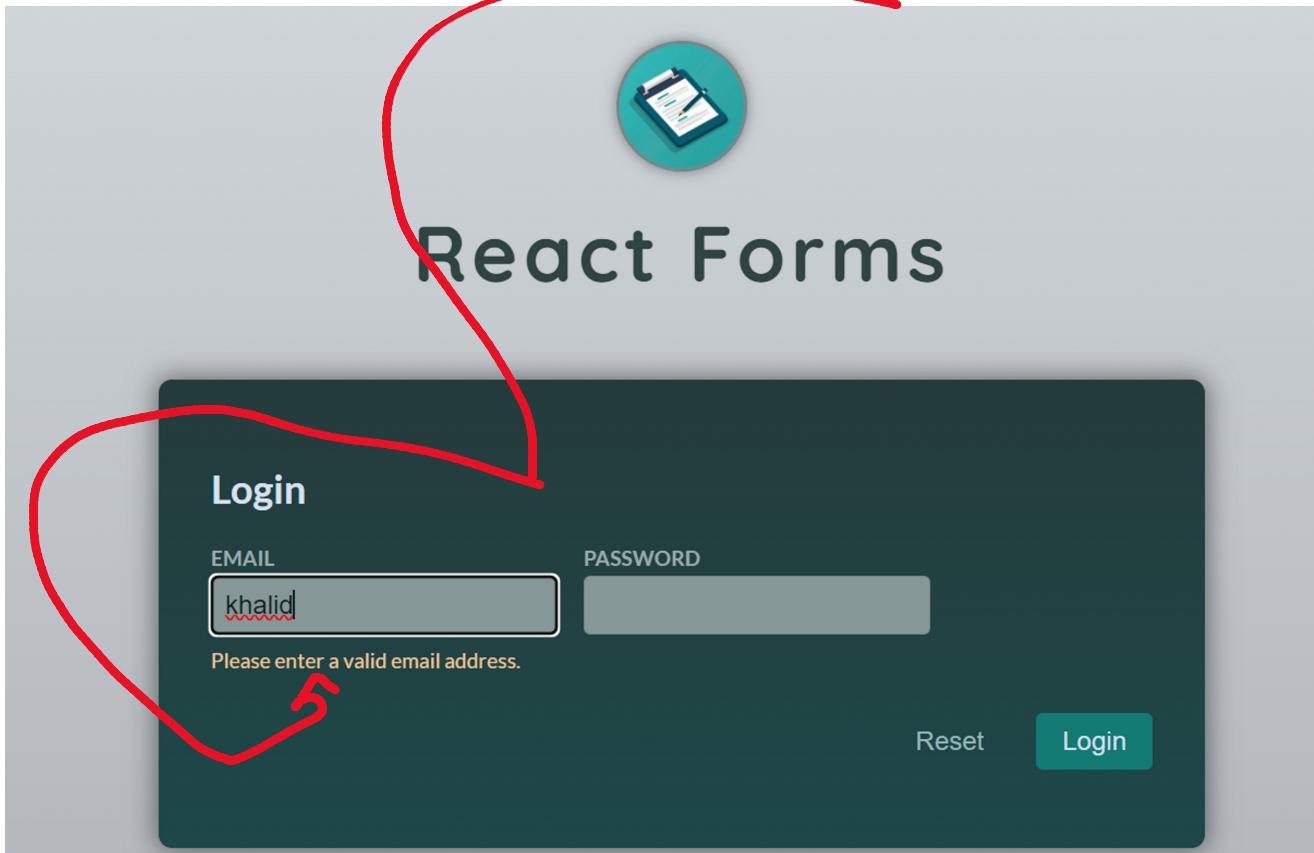
Validating Input on Every Keystroke via State

**Form Submission**

Handling submission is relatively **easy**
Entered values can be managed via **state**
Alternatively, they can be extracted via **refs**
Or via **FormData** and native browser features

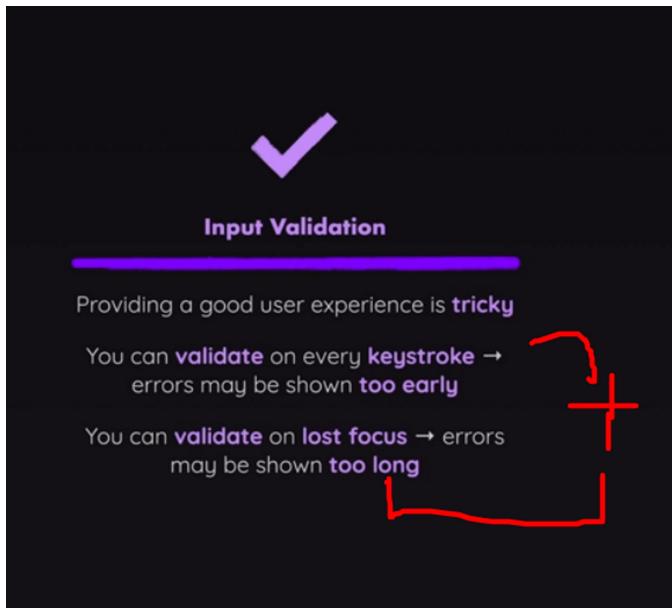
**Input Validation**

Providing a good user experience is **tricky**
You can **validate** on every **keystroke** →
errors may be shown **too early**

**Validating Input Upon Lost Focus (Blur)**

You can **validate** on **lost focus** → errors
may be shown **too long**

Email field theke type kore focus sorale error ashbe. Then abr type korte thakle o error thekey jabe.



So, we can combine on **every keystroke + last focus**

Using Third-Party Form Libraries

The screenshot shows a Google search results page with a dark theme. The search query 'react form libraries' is entered in the search bar. The results list includes:

- React Hook Form** - performance, flexible and extensible form ...
The best React form library that I have ever used while building a react app because of its utility and simplicity. It has a lot of useful tools and doesn't ...
- Related questions :**
 - What is the best React library for forms?
 - What can React form libraries be good for?
 - What is the best form library for next JS?
 - What is the best form builder in React?
- Formik** - Build forms in React, without the tears
Build forms in React, without the tears. Formik is the world's most popular open source form library for React and React Native.
- JavaScript in Plain English** - 3-best-react-form-li...
3 Best React Form Libraries Every React Developer ...

Feedback and Udemy logos are visible at the bottom right of the search results.