

Chapter 10: Xử lý Form và Validation trong React và Next.js

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Hiểu các phương pháp xử lý form trong React, bao gồm **controlled** và **uncontrolled components**.
- Sử dụng **React Hook Form** để xây dựng các form hiệu quả, giảm thiểu re-render và boilerplate code.
- Thực hiện **validation phía client** một cách dễ dàng và hiển thị thông báo lỗi thân thiện với người dùng.
- Xử lý trạng thái **loading** và **lỗi** khi submit form, bao gồm mô phỏng gọi API.
- Áp dụng các kỹ thuật tối ưu để cải thiện trải nghiệm người dùng (UX) và hiệu năng ứng dụng.

B. Nội dung lý thuyết

1. Controlled Components

Controlled Components là cách tiếp cận truyền thống trong React để xử lý form. Giá trị của các input được lưu trữ trong **React state** và được đồng bộ hóa thông qua các sự kiện như `onChange`.

Cách hoạt động:

- Mỗi input được gắn với một state thông qua thuộc tính `value`.
- Mỗi lần người dùng nhập dữ liệu, hàm `setState` được gọi, dẫn đến component re-render.
- Ví dụ:

```
import { useState } from 'react';

export default function ControlledForm() {
  const [formData, setFormData] = useState({ name: '', email: '' });

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

  return (
    <form>
      <input
        name="name"
        value={formData.name}
        onChange={handleChange}
      />
      <input
        name="email"
        value={formData.email}
        onChange={handleChange}
      />
    </form>
  );
}
```

Ưu điểm:

- Dữ liệu luôn đồng bộ với state, dễ kiểm soát và xử lý.
- Dễ dàng thực hiện validation hoặc logic phức tạp dựa trên state.

Nhược điểm:

- Gây re-render liên tục mỗi khi người dùng nhập, đặc biệt với form lớn.
 - Code dài dòng (boilerplate) do cần quản lý state và sự kiện cho từng input.
-

2. Giới thiệu React Hook Form

React Hook Form là một thư viện mạnh mẽ để xử lý form trong React, với triết lý **uncontrolled components** để giảm thiểu số lần re-render và tối ưu hiệu năng.

Triết lý:

- Sử dụng **ref** để theo dõi giá trị input thay vì state, giảm re-render.
- Chỉ re-render khi cần thiết (ví dụ: khi có lỗi validation hoặc submit).
- Cung cấp API đơn giản để xử lý form và validation.

Ưu điểm:

- **Hiệu năng cao:** Giảm số lần re-render, đặc biệt với các form lớn.
- **Code gọn gàng:** Giảm boilerplate so với controlled components.
- **Validation dễ dàng:** Tích hợp sẵn các quy tắc validation và thông báo lỗi.
- **Tích hợp tốt với Next.js:** Hoạt động mượt mà trong môi trường SSR và CSR.

Các khái niệm cốt lõi:

- **useForm** : Hook chính để khởi tạo form. Trả về các hàm và object như **register** , **handleSubmit** , **formState** .

```
import { useForm } from 'react-hook-form';  
const { register, handleSubmit, formState: { errors } } = useForm();
```

- **register** : Đăng ký input với form, cho phép theo dõi giá trị và validation.

```
<input {...register('name', { required: 'Tên là bắt buộc' })} />
```

- **handleSubmit** : Hàm bao bọc để xử lý submit, tự động chạy validation trước.

```
const onSubmit = (data) => console.log(data);  
<form onSubmit={handleSubmit(onSubmit)} />
```

- `formState.errors` : Object chứa các lỗi validation, dùng để hiển thị thông báo.

```
{errors.name && <p className="error">{errors.name.message}</p>}
```

Validation:

- Truyền các quy tắc validation trực tiếp vào `register` :

```
<input  
  {...register('email', {  
    required: 'Email là bắt buộc',  
    pattern: {  
      value: /^S+@S+$/i,  
      message: 'Email không hợp lệ',  
    },  
  })}  
>
```

- Các quy tắc phổ biến: `required` , `minLength` , `maxLength` , `pattern` , `min` , `max` .

3. Xử lý trạng thái Loading và Lỗi

Khi submit form, cần xử lý các trạng thái như **loading** (đang gửi dữ liệu) và **lỗi** (nếu API thất bại). Điều này giúp cải thiện UX và tránh các lỗi như submit nhiều lần.

Cách thực hiện:

- Sử dụng state để theo dõi `isSubmitting` .
- Vô hiệu hóa nút submit khi `isSubmitting` là `true` .
- Hiển thị thông báo hoặc hiệu ứng loading (ví dụ: "Đang gửi...").
- Sử dụng `setTimeout` để mô phỏng gọi API trong quá trình học.

Ví dụ:

```
import { useState } from 'react';
import { useForm } from 'react-hook-form';

export default function Form() {
  const { register, handleSubmit } = useForm();
  const [isSubmitting, setIsSubmitting] = useState(false);

  const onSubmit = async (data) => {
    setIsSubmitting(true);
    await new Promise((resolve) => setTimeout(resolve, 2000)); // Mô phỏng A
    setIsSubmitting(false);
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('name')} />
      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Đang gửi...' : 'Gửi'}
      </button>
    </form>
  );
}
```

C. Bài tập thực hành

Bài 1: Cài đặt React Hook Form

1. Cài đặt thư viện:

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

2. Kiểm tra cài đặt bằng cách import `useForm` trong một component bất kỳ:

```
import { useForm } from 'react-hook-form';
```

Bài 2: Tạo form liên hệ trong `pages/contact.js`

1. Tạo file `pages/contact.js` :

```
import { useForm } from 'react-hook-form';

export default function Contact() {
  const { register, handleSubmit } = useForm();

  const onSubmit = (data) => {
    console.log(data);
  };

  return (
    <div className="max-w-md mx-auto mt-10">
      <h1 className="text-2xl font-bold mb-4">Liên hệ</h1>
      <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
        <div>
          <label htmlFor="name" className="block">Họ tên</label>
          <input
            id="name"
            {...register('name')}
            className="w-full p-2 border rounded"
          />
        </div>
      </form>
    </div>
  );
}
```

```

    <div>
      <label htmlFor="email" className="block">Email</label>
      <input
        id="email"
        {...register('email')}
        className="w-full p-2 border rounded"
      />
    </div>
    <div>
      <label htmlFor="message" className="block">Tin nhắn</label>
      <textarea
        id="message"
        {...register('message')}
        className="w-full p-2 border rounded"
      />
    </div>
    <button
      type="submit"
      className="bg-blue-500 text-white px-4 py-2 rounded"
    >
      Gửi
    </button>
  </form>
</div>
);
}

```

2. Kiểm tra: Submit form và xem dữ liệu trong console.

Bài 3: Thêm validation cho form

1. Cập nhật `pages/contact.js` với các quy tắc validation:

```

import { useForm } from 'react-hook-form';

export default function Contact() {
  const { register, handleSubmit, formState: { errors } } = useForm();

  const onSubmit = (data) => {
    console.log(data);
  }
}

```



```

      {errors.message && <p className="text-red-500">{errors.message
    </div>
    <button
      type="submit"
      className="bg-blue-500 text-white px-4 py-2 rounded"
    >
      Gửi
    </button>
  </form>
</div>
);
}

```

2. Kiểm tra: Thử submit form với dữ liệu không hợp lệ (ví dụ: để trống hoặc nhập email sai định dạng).

Bài 4: Style thông báo lỗi

1. Cập nhật style cho thông báo lỗi trong `pages/contact.js` :

```

/* styles/globals.css */
.error {
  color: #ef4444;
  font-size: 0.875rem;
  margin-top: 0.25rem;
}

```

2. Đảm bảo các thông báo lỗi hiển thị bên dưới mỗi input với màu đỏ.

Bài 5: Mô phỏng submit form lên API

1. Cập nhật `pages/contact.js` để thêm trạng thái `isSubmitting` và mô phỏng API:

```

import { useForm } from 'react-hook-form';
import { useState } from 'react';

```

```

export default function Contact() {
  const { register, handleSubmit, formState: { errors } } = useForm();
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [submitSuccess, setSubmitSuccess] = useState(false);

  const onSubmit = async (data) => {
    setIsSubmitting(true);
    setSubmitSuccess(false);
    await new Promise((resolve) => setTimeout(resolve, 2000)); // Mô phỏng
    setIsSubmitting(false);
    setSubmitSuccess(true);
    console.log(data);
  };

  return (
    <div className="max-w-md mx-auto mt-10">
      <h1 className="text-2xl font-bold mb-4">Liên hệ</h1>
      {submitSuccess && (
        <p className="text-green-500 mb-4">Form đã được gửi thành công!<
      )}
      <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
        <div>
          <label htmlFor="name" className="block">Họ tên</label>
          <input
            id="name"
            {...register('name', { required: 'Họ tên là bắt buộc' })}
            className="w-full p-2 border rounded"
          />
          {errors.name && <p className="text-red-500">{errors.name.messa
        </div>
        <div>
          <label htmlFor="email" className="block">Email</label>
          <input
            id="email"
            {...register('email', {
              required: 'Email là bắt buộc',
              pattern: {
                value: /^\\S+@\\S+$/i,
                message: 'Email không hợp lệ',
              },
            })}
            className="w-full p-2 border rounded"
          />

```

```

        {errors.email} && <p className="text-red-500">{errors.email.mes
    </div>
    <div>
        <label htmlFor="message" className="block">Tin nhắn</label>
        <textarea
            id="message"
            {...register('message', {
                required: 'Tin nhắn là bắt buộc',
                minLength: {
                    value: 10,
                    message: 'Tin nhắn phải có ít nhất 10 ký tự',
                },
            })}
            className="w-full p-2 border rounded"
        />
        {errors.message} && <p className="text-red-500">{errors.message
    </div>
    <button
        type="submit"
        disabled={isSubmitting}
        className={`bg-blue-500 text-white px-4 py-2 rounded ${
            isSubmitting ? 'opacity-50 cursor-not-allowed' : 'hover:bg-b
        }`}
    >
        {isSubmitting ? 'Đang gửi...' : 'Gửi'}
    </button>
</form>
</div>
);
}

```

2. Kiểm tra:

- Nhấn submit và quan sát nút chuyển thành "Đang gửi..." và bị vô hiệu hóa.
- Sau 2 giây, kiểm tra thông báo thành công xuất hiện.

D. Bổ sung: Tối ưu hóa UX cho Form

1. Reset Form sau khi Submit

- Sử dụng `reset` từ `useForm` để xóa dữ liệu form sau khi submit thành công:

```
const { register, handleSubmit, reset, formState: { errors } } = useForm

const onSubmit = async (data) => {
  setIsSubmitting(true);
  await new Promise((resolve) => setTimeout(resolve, 2000));
  setIsSubmitting(false);
  setSubmitSuccess(true);
  reset(); // Xóa dữ liệu form
};
```

2. Xử lý lỗi API

- Mô phỏng lỗi API bằng cách thêm state `errorMessage` :

```
const [errorMessage, setErrorMessage] = useState('');


const onSubmit = async (data) => {
  setIsSubmitting(true);
  setErrorMessage('');
  try {
    await new Promise((resolve, reject) => setTimeout(() => reject(new E
  } catch (error) {
    setErrorMessage(error.message);
  }
  setIsSubmitting(false);
};

// Hiển thị lỗi
{errorMessage && <p className="text-red-500">{errorMessage}</p>}
```

3. Tăng trải nghiệm người dùng

- Thêm hiệu ứng loading (spinner) thay vì chỉ hiển thị “Đang gửi...”:

```
.spinner {  
  display: inline-block;  
  width: 20px;  
  height: 20px;  
  border: 2px solid #fff;  
  border-top-color: transparent;  
  border-radius: 50%;  
  animation: spin 1s linear infinite;  
}  
  
@keyframes spin {  
  to { transform: rotate(360deg); }  
}
```

```
<button  
  type="submit"  
  disabled={isSubmitting}  
  className={`bg-blue-500 text-white px-4 py-2 rounded flex items-center  
    isSubmitting ? 'opacity-50 cursor-not-allowed' : 'hover:bg-blue-600'  
  }`  
>  
  {isSubmitting ? (  
       
    <span className="spinner mr-2"></span> Đang gửi...  
  ) : (  
    'Gửi'  
  )}  
</button>
```

E. So sánh Controlled vs. Uncontrolled Components (React Hook Form)

Tiêu chí	Controlled Components	React Hook Form (Uncontrolled)
Hiệu năng	Re-render mỗi khi input thay đổi	Ít re-render, sử dụng ref để theo dõi input
Code	Dài dòng, cần quản lý state và sự kiện	Gọn gàng, ít boilerplate
Validation	Phải tự viết logic validation	Tích hợp sẵn validation, dễ cấu hình
Tích hợp API	Phải tự xử lý loading và lỗi	Dễ dàng tích hợp với API và xử lý trạng thái
Phù hợp	Form nhỏ, logic đơn giản	Form lớn, cần hiệu năng cao và validation phức tạp

F. Tài liệu tham khảo

- [React Hook Form Documentation](#)
- [Next.js Documentation - Forms](#)
- [React Documentation - Forms](#)

G. Bài tập nâng cao (tùy chọn)

- Tích hợp thư viện validation:** Sử dụng thư viện `yup` kết hợp với `react-hook-form` để validation form phức tạp hơn (ví dụ: password confirmation).

```
npm install @hookform/resolvers yup
```

```
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';

const schema = yup.object().shape({
  name: yup.string().required('Họ tên là bắt buộc'),
  email: yup.string().email('Email không hợp lệ').required('Email là bắt buộc'),
  message: yup.string().min(10, 'Tin nhắn phải có ít nhất 10 ký tự').required('Tin nhắn là bắt buộc');
});

const { register, handleSubmit, formState: { errors } } = useForm({
  resolver: yupResolver(schema),
});
```

2. **Tùy chỉnh thông báo lỗi:** Hiển thị thông báo lỗi dưới dạng tooltip thay vì văn bản bên dưới input.
3. **Gọi API thực tế:** Thay `setTimeout` bằng một API thực tế (ví dụ: `fetch` đến `jsonplaceholder.typicode.com`). Xử lý cả trường hợp thành công và thất bại.
4. **Form multi-step:** Tạo một form liên hệ nhiều bước (bước 1: thông tin cá nhân, bước 2: tin nhắn, bước 3: xác nhận) sử dụng React Hook Form.