Implementing Form Validation

Cory House
REACT CONSULTANT AND TRAINER

@housecor   reactjsconsulting.com

Demo

Build checkout and implement validation

- Validate onBlur and onSubmit
- Track touched fields
- Derive most validation-related values
- Implement a state enum

Form Validation Decisions

Validation Decisions

1	**Where to display errors** By field, at top	By field and at top
2	**When to display errors** onSubmit, onBlur, onChange	All three
3	**When to disable submit** Until clean, submitting, never	While submitting
4	**When to revalidate** onSubmit, onChange, onBlur	All three

Goals

- ✓ Display error summary at top on submit.
- ✓ Validate onBlur. Display error message next to field at that time.
- ✓ Submit button should be disabled when save is in progress.
- ✓ Revalidate onChange

What State Do We Need?

Store as “touched”	touched	What fields have been touched?
Store as “status”	submitted	Has the form been submitted?
	isSubmitting	Is a form submission in progress?
Derive	isValid	Is the form currently valid?
	errors	What are the errors for each field?
	dirty	Has the form changed?

What State Do We Need?

Store as “touched”	touched	What fields have been touched?
Store as “status”	submitted	Has the form been submitted?
	isSubmitting	Is a form submission in progress?
Derive	isValid	Is the form currently valid?
	errors	What are the errors for each field?
	dirty	Has the form changed?

State Enums

Favor “enums” over separate booleans

```
// Using separate state to track the form's status: (risk of out-of-sync) 👎
const [submitting, setSubmitting] = useState(false); // Submit in progress
const [submitted, setSubmitted] = useState(false);   // Submitted with errors
const [completed, setCompleted] = useState(false);   // Completed



// Using a single status “enum” instead 👍
const STATUS = {
  IDLE: "IDLE",
  SUBMITTING: "SUBMITTING",
  SUBMITTED: "SUBMITTED",
  COMPLETED: "COMPLETED",
};

const [status, setStatus] = useState(STATUS.IDLE);
```

State Enums vs Finite State Machines

Does my logic have discrete states?

If so, consider declaring a single “status” variable

Finite State Machine

Only *one* state can be active at the same time. The machine transitions from one state to another.

XState

Open source Finite State Machine

Key benefits over simple state enums:

1. **Enforce state transitions**
   - Declare how and when your app moves between states
   - Protects from invalid transitions

State Enums

Favor “enums” over separate booleans

```
// Using a single status “enum”
const STATUS = {
  IDLE: "IDLE",
  SUBMITTING: "SUBMITTING",
  SUBMITTED: "SUBMITTED",
  COMPLETED: "COMPLETED",
};

const [status, setStatus] = useState(STATUS.IDLE);
```

XState

Open source Finite State Machine

Key benefits over simple state enums:

1. **Enforce state transitions**
   - Declare how and when your app moves between states
   - Protects from invalid transitions
2. **State charts**

Form Libraries

Formik

React Hook Form

Summary

Validate onBlur and onSubmit

Tracked touched fields

Derived most validation-related values

Implemented state enum pattern

Consider XState for finite state machines

Check for errors “on-the-fly”

Honor Principle of Least Privilege

- Try to pass only what’s needed

Next up: Managing state via Refs