Implementing Data Validation

Nitin Singh
Full-stack Developer

Overview

Understand different flavors of validations supported by Formik

How to add form level validations

How to add field level validations

Configuring triggers for running validations

How to manually trigger validations

Using Yup for schema-based validations

Implementing Form Validation with Formik

Flavors of Validation

1

Form level validation

Specified on the <Formik> tag

2

Field level validation

Specified on the <Field> tag

Form Level Validation

Specified using validate, validationSchema props on <Formik> tag

Provides access to all values of the form and returns an error object map with field names as keys

Validation function can be sync or async. Async functions return a Promise

Especially useful when validating dependent fields

Field Level Validation

Specified using validate prop on the <Field> tag

Provides access to the value for the field for validation and returns a string as error

Validation function can be sync or async. Async functions return a Promise

Only mounted fields are validated

Bugs, Bugs, Everywhere!

If no error is present for a field, return undefined instead of null. If you return null then computed Formik properties like isValid won’t work as expected.

Validation Triggers

1 On blur events/methods

2 On change events/methods

3 On <Formik> mount / initial value changes

4 When form submission is attempted

Controlling Validation Trigger Behaviour

1

On Blur

Default value true. Configurable using validateOnBlur prop on <Formik> tag

2

On Change

Default value true. Configurable using validateOnChange prop on <Formik> tag

3

On Mount

Default value false. Configurable using validateOnMount prop on <Formik> tag

Manually Triggering Validations

Use validateForm() method for form level validation

Use validateField(fieldName) method for field level validation

Use isValidating() method to check whether validation is already running

Displaying Validation Errors

Use the <ErrorMessage> component

It will only show errors if a field has been touched

Allows total customization via render and component props

Demo

Invoice tracker application

Add Sign-up form

Add form level validations for all fields

Add field level validation for password field to display strength of input password

Adding Schema-based Validations Using Yup

Why Schema-based Validation

1

No repetition

Avoid writing repetitive checks for things like strings being empty

2

Consistent checks

Consistent logic for validating things like email

3

Concise, readable code

Schema defines the requirements on a field very concisely

Using Yup with Formik

1. To install Yup in your project run: npm install yup --save

2. Specify the form level schema using the validationSchema prop on the <Formik> tag

3. Use validate or validateSync methods on schema to manually trigger validations for field level validations

```
const schema = Yup.object().shape({

   name: Yup.string()
   .min(2, 'Too Short!’)
   .max(50, 'Too Long!’)
   .required('Required’),

   email: Yup.string()
   .email('Invalid email’)
   .required('Required'),

 });
```

◂ Create Yup object schema

◂ Specify that name is a string

◂ Specify min, max length for name

◂ Specify that the field can’t be empty

◂ Specify that email is a string

◂ Check that the email string is a valid email address

Salient Features of Yup

Method chaining to make the code as concise as possible

Allows conditional checking for dependent fields using the when method

A ton of helpers for all data types

Demo

Invoice tracker application

Rewrite validations for the Sign-up form using Yup

Summary

Understanding various types of validations supported by Formik

Adding form level validations

Adding field level validations

Understanding trigger points for running validations

Running validations manually

Integrating schema-based validations using Yup

Up Next:
Creating Reusable Custom Form Elements