Creating Forms Using Formik

Nitin Singh
Full-stack Developer

Overview

Disadvantages of using vanilla React for creating forms

Understanding how Formik overcomes these problems

Components provided by Formik

Understanding how Formik maintains input and error states for fields

Helpers provided by Formik for common operations

Problems with Using Vanilla React for Forms

Very verbose

Repetitive code for maintaining state and handling callbacks

Error prone

Formik is a library that solves all the above problems with writing controlled React forms.

Salient Features of Formik

Automatically keeps track of values, errors and visited fields

Automatically hooks up appropriate callback functions on form elements

Provides helpers for sync and async validations and showing error messages

Provides sensible defaults for common functionalities while allowing maximum customization

Setting up Formik in Your App

1. **To install formic run: npm install formik --save**

2. **You don’t need to migrate all forms in your app at once**

3. **Replace the form and form-element tags in your components with respective Formik components**

Key Formik Components

1 <Formik>

2 <Form>

3 <Field>

4 <ErrorMessage>

```
<Formik >
   <Form>
     <label htmlFor="firstName">First Name</label>
     <Field name="firstName" type="text" />
     <ErrorMessage name="firstName" />
     <button type="submit">Submit</button>
   </Form>
</Formik>
```

Anatomy of a Formik Form

<Formik> Component

This is the top-level component which controls all other components

Responsible for keeping track of state for the field components using their “name” attribute as key

Responsible for injecting and handling callbacks for field components

Provides properties to set initial values, validation function and handle callbacks like submit and reset

Allows rendering via children and custom components

How Formik Maintains State for Form Fields

Formik maintains three maps to automatically track the state of all the <Field> tags it contains.

values

Map of the <Field> tags' name to its user input value

errors

Map of the <Field> tags' name to the error string returned by validation functions

touched

Map of the <Field> tags' name to a boolean indicating whether it was ever focused or not

<Form> Component

Wrapper around HTML <form> tag

Automatically hooks parent Formik's on submit and reset callbacks into the HTML form

Extra props are passed to the DOM directly

<Field> Component

- By default it renders a HTML <input> tag
- Uses the name attribute on the <Field> to match up with the Formik state
- <Formik> will automatically inject onChange, onBlur and value attributes for all <Field> tags
- To render a non-input tag set the “as” attribute of the <Field> tag to textarea, select or any custom component
- Allows for field level validation via the validate attribute

<ErrorMessage> Component

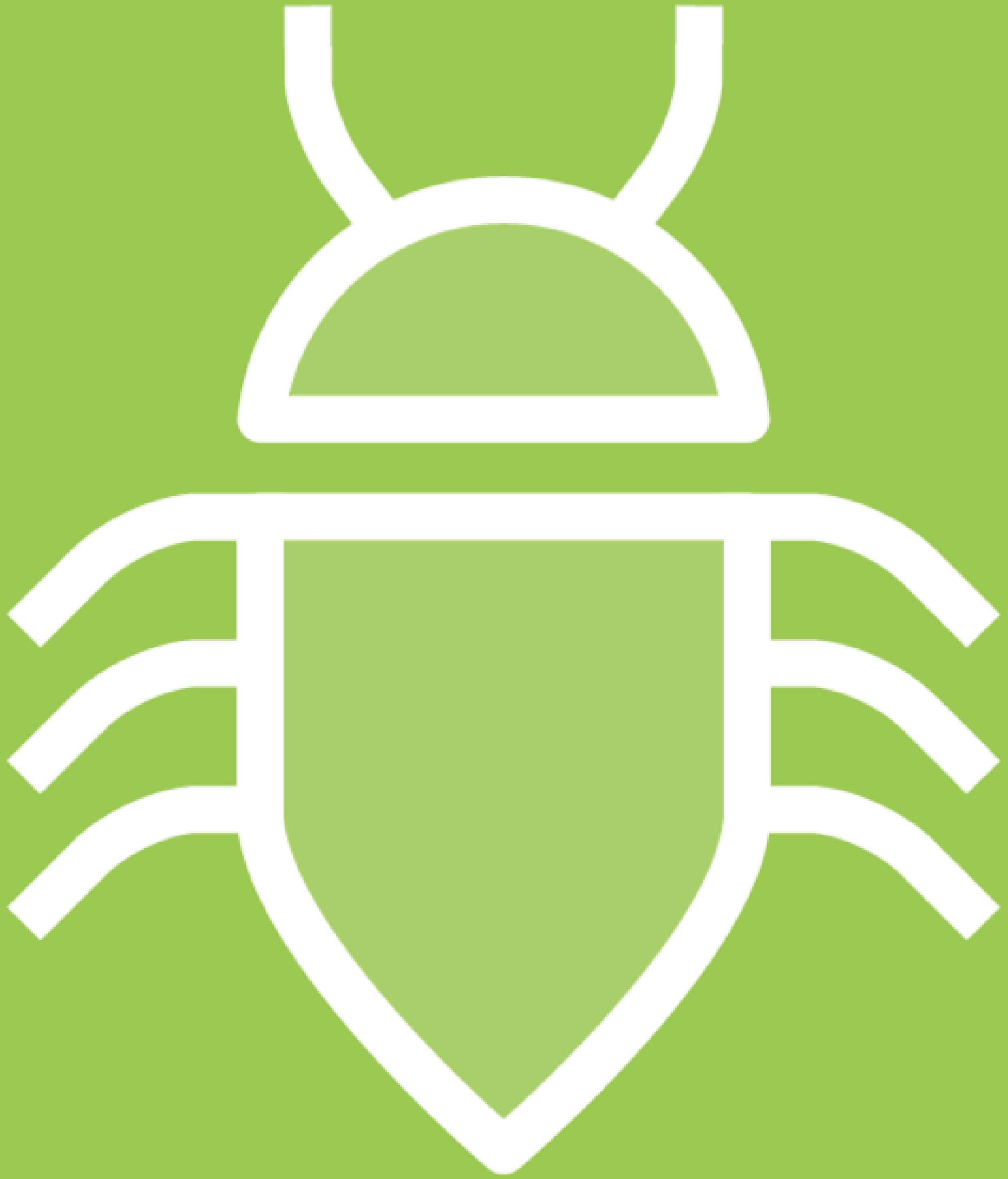
Represents error message for the <Field> tag whose “name” attribute matches with its own “name” attribute

The message to render is always expected to be a string

Fully custom rendering supported via children, component and render properties

```
<Formik

   initialValues={{ username: 'jared' }}

   validate={(values, props) => {

     // ..validation logic

   }}

   onSubmit={(values, actions) => {

      const uname = values.username

   }}>

 {props => (
  <Form>
     <Field name="username" type="text" />
  </Form>
 )}

</Formik>
```

- Set initial values for the Fields as a JSON object with <Field> tag's name as key
- Business logic to validate input and return an error JSON object with <Field> tag's name as key
- Handle form submission
- Use the props variable to access Formik's helper functions
- <Field> tag that maps to <input> tag of type text



Bugs, Bugs, Everywhere!

Even if your form is empty by default, you must initialize all fields with initial values. Otherwise React will throw an error saying that you have changed an input from uncontrolled to controlled.

Key <Formik> Helper Props

1 **isSubmitting**

Indicates if form is under submission

2 **setSubmitting(bool)**

Set a form's submission phase

3 **isValid**

Indicates if form input is valid

4 **isValidating**

Indicates if form is being validated

5 **resetForm(values)**

Reset form using given values

Handling Form Submission

On form submission the onSubmit callback on the <Formik> tag is invoked with the field values and validations are run

The onSubmit can be sync or it can be async and return a Promise

In sync onSubmit call setSubmitting(false) to indicate end of submission

Demo

Invoice tracker application

Install Formik in the project

Rewrite Sign In form using Formik

Add validation for all fields as earlier

Disable submit button post click using Formik helpers and make API call

Summary

Downsides of using vanilla React to create forms

Salient features of Formik that help write concise form components

Various components provided by Formik and their properties

How Formik maps fields to values, errors automatically

Most useful helpers provided by Formik for common form operations

Up Next:
Implementing Data Validation