Integrating with form libraries

UDS offers a variety of input components that can be used to build custom forms for your application. That said, most of the time developers will probably want to use an third-party library to handle form state. Below is a brief overview of different aspects of integrating input components with some of the most popular form libraries within the React ecosystem.

Formik

Formik is arguably one of the most popular libraries used by React developers to handle forms. It's relatively easy to adopt and it provides a way to use your own input components by rendering them via a stateful wrapper that utilizes `useField` hook for state management:

```
import { Formik, Form, useField } from 'formik'

const MyTextInput = ({ label, ...rest }) => {
  const { name } = rest
  const [field, meta] = useField(name)
  const { onChange } = field
  const handleChange = (value) => onChange({ target: { name,
value } })

  return (
    <TextInput
      label={label}
      {...(meta.touched && meta.error && { validation: 'error',
feedback: meta.error })}
      {...field}
      {...rest}
      onChange={handleChange}
    />
  )
}
```

Then you can render your form using the following code:

```
const FormComponent = () => (
  <Formik
```

```
    initialValues={{
      firstName: '',
      lastName: ''
    }}
    onSubmit={(values, { setSubmitting }) => {
      setTimeout(() => {
        alert(JSON.stringify(values, null, 2))
        setSubmitting(false)
      }, 400)
    }}
  >
    <Form>
      <MyTextInput label="First Name" name="firstName"
id="firstName" />
      <MyTextInput label="Last Name" name="lastName"
id="lastName" />
      <button type="submit">Submit</button>
    </Form>
  </Formik>
)
```

Alternatively, you can use `useFormik` hook if you for some reason prefer not to create wrappers.

React Hook Form

React Hook Form became incredibly popular in recent years. It reduces the amount of code you need to write while enhancing its performance by removing unnecessary re-renders.

The easiest way to use UDS components with React Hook Form is via the `useForm` hook:

```
import { BaseProvider, TextInput } from '@telus-uds/components-
web'
import alliumTheme from '@telus-uds/theme-allium'
import { useForm } from 'react-hook-form'

const FormComponent = () => {
  const { register, handleSubmit } = useForm()
  const onSubmit = async (data, event) => console.log(data,
event)
```

```
  const handleChange =
    ({ name, onChange }) =>
    (value) =>
      onChange({ target: { name, value } })
  const firstName = register('firstName', {
    required: 'Please enter your first name.'
  })
  const firstNameProps = {
    label: 'First Name',
    ...firstName,
    onChange: handleChange(firstName)
  }
  const lastName = register('lastName', {
    required: 'Please enter your last name.'
  })
  const lastNameProps = {
    label: 'Last Name',
    ...lastName,
    onChange: handleChange(lastName)
  }

  return (
    <BaseProvider theme={alliumTheme}>
      <form onSubmit={handleSubmit(onSubmit)}>
        <TextInput {...firstNameProps} />
        <TextInput {...lastNameProps} />
        <button type="submit">Submit</button>
      </form>
    </BaseProvider>
  )
}
```

React Hook Form also provides a `Controller` wrapper for controlled components that can be used with UDS inputs as well:

```
import { useForm, Controller } from 'react-hook-form'

const FormComponent = () => {
  const onSubmit = async (data, event) => console.log(data,
event)
  const handleChange =
    ({ name, onChange }) =>
    (value) =>
      onChange({ target: { name, value } })
```

```
  return (
    <BaseProvider theme={alliumTheme}>
      <form onSubmit={handleSubmit(onSubmit)}>
        <Controller
          name="firstName"
          control={control}
          render={({ field }) => (
            <TextInput
              label="First Name"
              value={field.value ?? ''}
              onChange={handleChange(field)}
            />
          )}
        />
        <Controller
          name="lastName"
          control={control}
          render={({ field }) => (
            <TextInput label="Last Name" value={field.value ??
''} onChange={handleChange(field)} />
          )}
        />
        <button type="submit">Submit</button>
      </form>
    </BaseProvider>
  )
}
```

In any case, the most important thing to remember is that unlike on the UDS inputs themselves, `onChange` handler here (as well as in the case of Formik) accepts the event object as the first argument and looks for a field `name` and the `value` within this event.

React Final Form

React Final Form is a high performance subscription-based form state management for React (and also a successor of Redux Form). Its API contains both component-based and hook-based tools that can be used to create forms of various levels of complexity. The simplest way to use UDS components with React Final Form is via render functions:

```
import { BaseProvider, TextInput } from '@telus-uds/components-web'
import alliumTheme from '@telus-uds/theme-allium'
import { Form, Field } from 'react-final-form'

const FormComponent = () => {
  const onSubmit = (data) => console.log(data)
  const handleChange =
    ({ name, onChange }) =>
    (value) =>
      onChange({ target: { name, value } })

  return (
    <BaseProvider theme={alliumTheme}>
      <Form
        onSubmit={onSubmit}
        render={({ handleSubmit }) => (
          <form onSubmit={handleSubmit}>
            <Field
              name="firstName"
              render={({ input, meta }) => (
                <TextInput
                  label="First Name"
                  value={input.value ?? ''}
                  onChange={handleChange(input)}
                  {...(meta.touched &&
                    meta.error && {
                      validation: 'error',
                      feedback: meta.error
                    })}
                />
              )}
            />
            <Field
              name="lastName"
              render={({ input, meta }) => (
                <TextInput
                  label="Last Name"
                  value={input.value ?? ''}
                  onChange={handleChange(input)}
                  {...(meta.touched &&
                    meta.error && {
                      validation: 'error',
                      feedback: meta.error
                    })}
```

```
                />
              )}
            />
            <button type="submit">Submit</button>
          </form>
        )}
      />
    </BaseProvider>
  )
}
```

Keep in mind that you can also use those render functions as children of the `Field` component, as well as the hook-based API (which is what `Form` and `Field` components are using under the hood).

Edit this page