# React Forms

Different approaches to dealing with forms

## Our goals

- Understand the patterns used for dealing with forms
- Using the ref pattern for component communication
- Higher Order component pattern
- Render props pattern
- Formik
- Grabbing the input from the forms, handling validations, and handling errors

## Form pattern - Controlled component - EX

- With controlled form control the value is determined by the state
- We attach an event to the input change and set the state with the new value in the event handler
- We can validate the input before placing in the state and make sure our state always contains a validated input
- Create an input to enter your name
- Make sure that value is always capitalised

## children props

- this.props.children
- special property that can be passed like regular property or between the tags of your react component

## React Pattern - Render Props

- Used for sharing behaviour between components
- You create a parent component that holds the common behaviour you want to share
- In the parent component you get a property of a function which will return a react component or elements
- That function you call in the parent render method usually passing arguments to the function used to connect the common behaviour to the items rendered in the function
- That function sometimes is passed through the props.children
- EX: encapsulate the handleChange in the controlled form using this pattern

## pattern - Higher Order Components - EX

- HOC is a decorator pattern for react component to add additional power to a component
- Essentially it is a function that gets a component and returns “better” component
- It is often common to create a factory method that returns a function that gets a component (for example react-redux connect)
- Lets build a HOC component that logs the properties a component recieves

## ref

- We use ref as a form of communication from parent to child
- ref gives us access to DOM element or component instance
  - can not be used on function components
- This means the parent can call methods on the child instance class
- you create it usually in the constructor with React.createRef
- You pass it to the ref property of the dom element or component instance you want to reference in the render method
- The ref property will be assigned before componentDidMount / componentDidUpdate

## Forms pattern - Uncontrolled form control

- With this form pattern you grab the form data only when you need to use it (usually in the form submit)
- you grab the data by placing ref in all the inputs you need to grab

## pattern - ref forwarding

- We use this pattern when our parent needs to pass ref to child but we expect the ref to be populated with an inner dom element or inner component class in the child
- React.forwardRef uses the Higher Order Components pattern
- It will get a function with props and ref as arguments and will have to return React.createElement where we pass the ref we got in the arguments to the inner part of the child component
- This pattern will allow us to use the ref, we can also avoid using the React.forwardRef and just use a different property name to pass the React.createRef()

## Formik

- library that helps us solve common form problems
- The library provides us with a Formik component that wraps our form
- The library works with the render props pattern so we need to supply the library with a render function through children.props that will render our form
- The render function will be called with properties that we need to attach to our form and form inputs

## Formik - properties

- We need to supply Formik with the following properties
  - children - containing our form render function
  - initialValues
  - validate - provide a function that gets the values and will return error object or null
  - onSubmit - will be called by Formik passing values, actions

## Formik - render function

- We need to supply Formik with the render function either using the render prop or using the children prop
- Our render function callback will be called with the following arguments
  - values - connect me to the inputs
  - errors - use me to display errors
  - touched
  - handleBlur - connect to each input blur
  - handleChange - connect to each input change
  - handleSubmit - connect to the form submit event
  - isSubmitting - the form is currently processed

# Formik - Shortcuts

- Instead of always connecting the form and input you can use Formik shortcut components that will already be connected
- Form - will connect to the submit and reset
- Field - Will connect an input to the function, only need name
- ErrorMessage - Will connect to the proper error, only needs a name

# Validation - Yup

- Library for creating validation schema object
- You start with create an object using yup.object()
- You call shape({ …}) on that object to define how your object looks like
- The key will match your formik input names, and the value need to match how the proper value of that field looks like
  - yup.string().email(‘Email format is bad’)
- you can create custom validation for field with:
  - Yup.string().test(‘test-name’, ‘custom error message’, function(value) {return true / false})
  - You can return Promise as well

## Formik + Yup for validation - EX

- You can provide Formik with the property validationSchema which will accept Yup schema
- Let’s try and create a login form with email and password and validate that the email is correct and the password is more that 5 characters length

# Formik Dealing with errors - EX

- In the handleSubmit method we will use the actions.setErrors to set error we get from our api
- Make sure you display them in your form

## Summary

- Know React patterns and you will very quicky understand other React libraries
- Understand the render props means understanding how to use Formik
- Understanding the HOC pattern means you will understand the withFormik method
- Using patterns we solved common forms problem and have the proper tools to solve many other repeating problems