Virtual DOM makes forms in react much different from forms in other frameworks like angular, knockout and ember.

React has some surprising ways of handling forms

ControllerView

Critical where forms are concerned

Each component’s render must specify a single top-level component

BAD

render: function() {

return (

<h1>Manage Author</h1>

<AuthorForm/>

);

}

GOOD

render: function() {

return (

<div>

<h1>Manage Author</h1>

<AuthorForm/>

</div>

);

}

## Controlled components

Any input that has a value is a controlled component

Any controlled component with a value must have a changehandler otherwise the user’s input won’t register.

Uncontrolled components will act just like inputs you’re used to.

Inputs won’t accept user provided data until handlers are attached

Input is processed internally by react

When it notices there are no handlers, it ignores input and doesn’t update the view / component

React requires explicitly setting up change handlers on input controls

View controllers are the best place to handle state changes

There’s somewhat more typing required to get things done in react in comparison to other frameworks

You need to attach handlers to all inputs for anything in the view to get updated.

Having a controller component allows child components to be parameterized for reuse.

JSX Notes

Attributes

Placeholder

Shadow text to put in an empty form field

Ref

For obtaining reference to a child component

Passed to React.findDomNodes

Validation

Ridirects

## Reusable inputs

Like JSP tag libraries

Reduces duplication

Enforces conventions

Managing bootstrap styles can be done by hand or more elegantly using various github projects

Like establishing an element’s classname, accounting for whether or not an error state is effective.

When developing reusable components, it’s helpful to use propTypes to ensure inputs are being used properly.

## Javascript

event.preventDefault

Prevent default browser behavior

Prevent browser from doing anything by default when an event is raised

## Bootstrap

Requires decorating markup with various classes and adding some markup structure you might not need otherwise.

User notifications

Saving and Populating data on load

When and where to wire these up

Best practices

## Programmatic redirects

React-router supports this

Navigation mixin

## Notifications

Toaster library

Comprised of a js library and css

Like other libs (bootstrap), you want to bundle the css in your app via gulp.

## Validation

Form validation

## PropTypes

Validate data being passed to components

Allows you to harden expectations that a component is making

Declare expectations about the component

Useful where a component will be reused by others

## Transitions

Prevent users from losing data when navigating away from a form

Statics property

willTransitionFrom(transition, component)

Component.state

Transition.abort

## Form hydration via URL

Given <http://nasdaq.com/author/mikeg>

This should result in a form filled in with mikeg’s data.

We want to click on a link and be able to see specific data, or edit that data, etc.

A good place to put logic to populate a form using url data is in componentWillMount

React puts url parameters into props.params

If you have url <http://foo.com/author/foo?test=123>

… you can access test using this.props.params.test

componentWillMount

Changing state in this method will not trigger a component to rerender

componentDidMount

Changing state in this method \_WILL\_ trigger a component to render

## Terms

2-way data binding

Changes to data model get propogated to UI

Changes to UI get propogated to the model

Unidirectional data binding

UI is only updated in response to changes in the model

Mounting

TODO