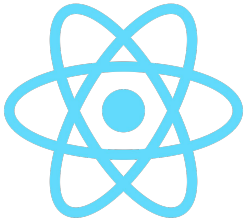


# Unit Testing React & Redux

Dylan Kirby

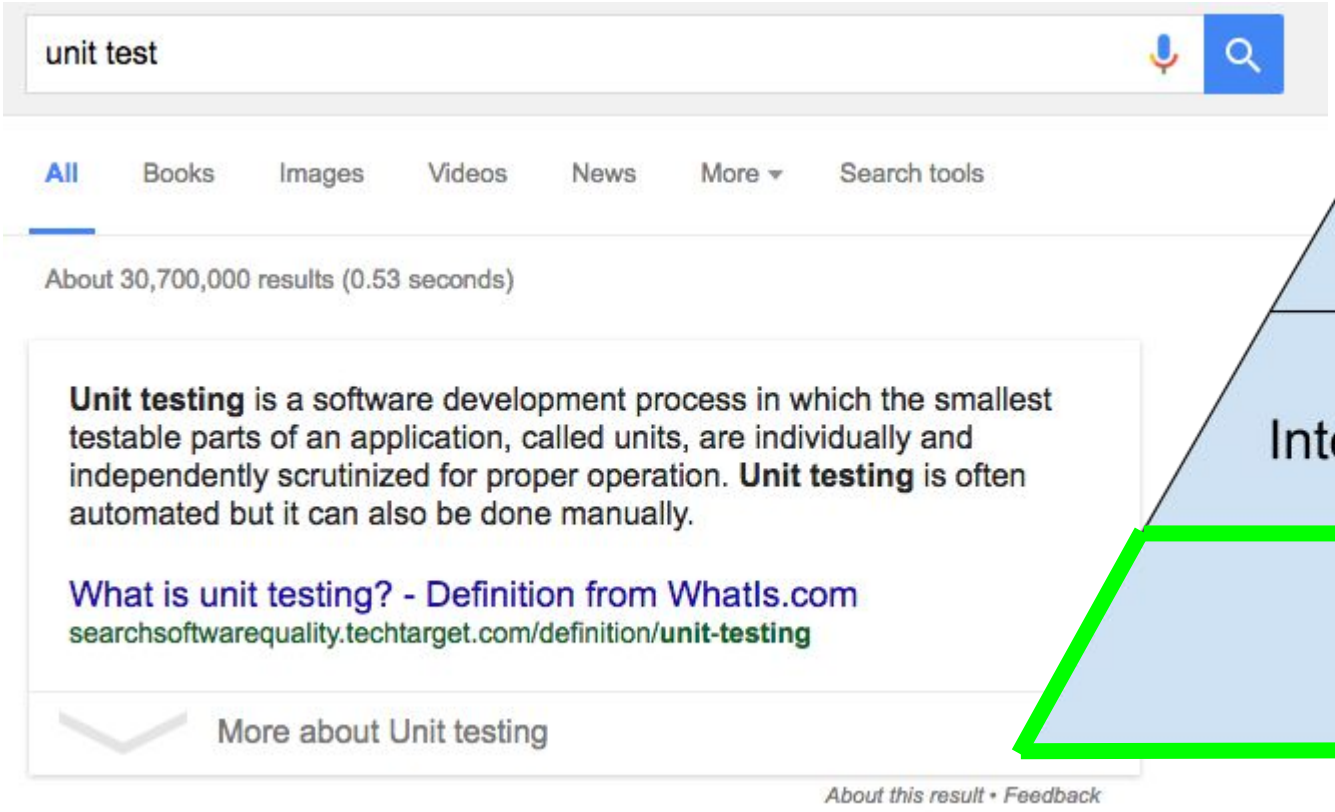
Slides: github.com/djkirby/react-symposium-unit-testing

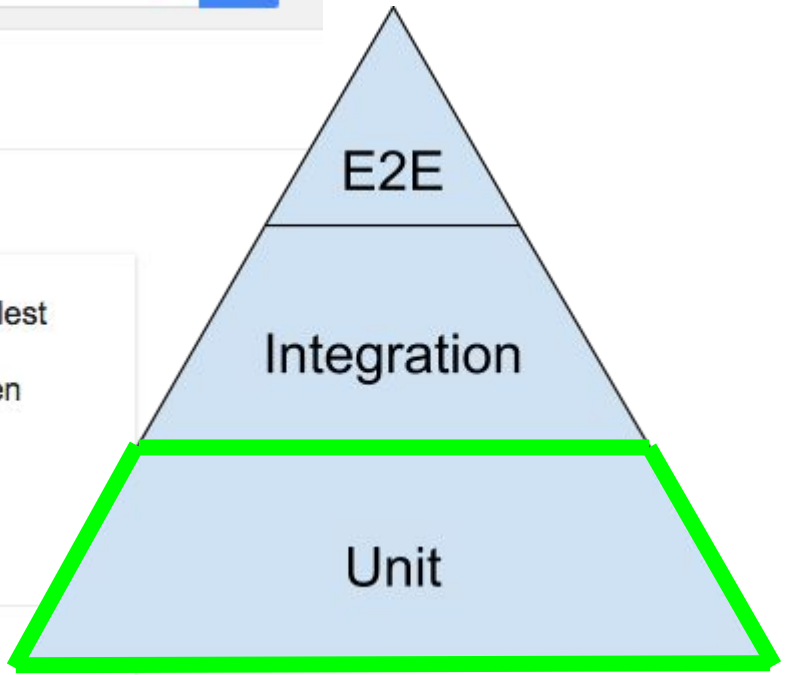
# Overview

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

# What is Unit Testing



# Why unit test?

- Check our code behaves as we expect it to
- Check our code handles edge cases/errors correctly
- Check our code is syntactically correct (especially in a dynamic language)
- Reduce dependency on end-to-end tests - faster, more accurate
- Tests act as "executable documentation" for our code
- Tests enable refactoring with confidence
- Tests can help us write better code

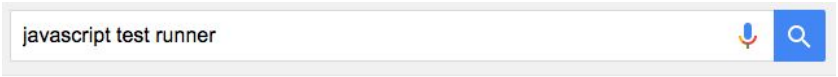
April 2016 London React User Group

Red Badger

Subscribe 500

https://www.youtube.com/watch?v=kkqArkYV4fw

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources




```
describe('Array#indexOf', () => {
  const arr = [1, 2, 3];
  it('should return -1 when the value is not present', () => {
    expect(arr.indexOf(5)).to.equal(-1);
    expect(arr.indexOf(0)).to.equal(-1);
  });
});
```

Back in the terminal:

```
$ mocha

  .

  ✔ 1 test complete (1ms)
```

# setup -> exercise -> verify -> teardown

```
describe('Array#length', () => {
  const myList = [1, 2, 3, 4];  // Setup
  const count = myList.length;  // Exercise
  it('returns the number of items in the array', () => {
    expect(count).to.equal(4);  // Verify
  });
});
```

http://www.agile-code.com/blog/the-anatomy-of-a-unit-test/

# chai

## Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.length(3);
tea.should.have.property('flavors')
  .with.length(3);
```

Visit Should Guide

## Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.length(3);
expect(tea).to.have.property('flavors')
  .with.length(3);
```

Visit Expect Guide

## Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

# Testing redux action creators

```
const addTodo = text => ({ type: 'ADD_TODO', text });

describe('add todo action', () => {
  it('should create an action to add a todo', () => {
    const action = addTodo('Give presentation');
    const expectedAction = { type: 'ADD_TODO', text: 'Give presentation' };
    expect(action).to.deep.equal(expectedAction);
  });
});
```

# Testing redux reducers

```
const todos = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, action.text];
    default:
      return state;
  }
};

describe('todos reducer', () => {
  it('should return the initial state', () => {
    const newState = todos(undefined, {});
    expect(newState).to.deep.equal([]);
  });

  it('should handle ADD_TODO', () => {
    const action = { type: 'ADD_TODO', text: 'Mow the lawn' };
    const newState = todos(['Give presentation'], action);
    expect(newState).to.deep.equal(['Give presentation', 'Mow the lawn']);
  });
});
```

```
it('should handle many actions', () => {
  const actions = [
    { type: 'ADD_TODO', text: 'foo' },
    { type: 'ADD_TODO', text: 'bar' },
    { type: 'ADD_TODO', text: 'baz' }
  ];
  const newState = actions.reduce(todos, []);
  expect(newState).to.deep.equal(['foo', 'bar', 'baz']);
});
```

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

# Unit Testing React Components

- Check it renders as expected given certain props/state
- Check it handles interactions as expected

- React addons test utils
- Enzyme

# React Addons Test Utils

- Simulate events
- Render component into DOM
- Shallow render

# Enzyme

- JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output
- Uses React test utilities, provides different interface to reduce boilerplate
- jQuery-like syntax
- Shallow rendering, full DOM rendering
- Guides for using w/ different test runners

# chai-enzyme

Chai.js assertions and convenience functions for testing React Components with enzyme

```
expect(wrapper.find(Button)).to.have.length(1);
=>    expect(wrapper).to.have.exactly(1).descendants(Button);

expect(wrapper.contains('Hello World')).to.be.true
=>     expect(wrapper).to.contain('Hello World');

expect(wrapper.find('.my-button').hasClass('disabled')).to.be(true);
=>     expect(wrapper).to.have.className('disabled');
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(selector)
contains(nodeOrNodes)
context([key])
**debug()**
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
find(selector)
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```jsx
const wrapper = shallow(
  <Book
    title="Huckleberry Finn"
    cover={{
      url: 'http://some.url/to/img.png',
      width: 40,
      height: 80
    }}
  />
);
console.log(wrapper.debug());
```

Outputs to console:

```jsx
<div>
 <h1 className="title">Huckleberry Finn</h1>
 <BookCover cover={{...}} />
</div>
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(**selector**)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(**selector**)
everyWhere(predicate)
filter(**selector**)
filterWhere(predicate)
find(**selector**)
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
not(**selector**)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
shallow([options])
simulate(event[, data])
some(**selector**)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
// css selectors
wrapper.find('.todo-list')
wrapper.find('button')
wrapper.find('div#some-id')

// imported component
wrapper.find(TodoList)

// component displayName
wrapper.find('TodoList')
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(selector)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
**find(selector)**
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
**prop([key])**
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
expect(
  wrapper.find('Thumbnail').prop('path')
).to.equal('foo.jpg');
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(selector)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
find(selector)
findWhere(predicate)
first()
forEach(fn)
get(index)
**hasClass(className)**
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
expect(wrapper).to.have.className('active');
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(selector)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
**find(selector)**
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
**text()**
type()
unmount()
update()

```
expect(
  wrapper.find('.cart-count').text()
).to.equal('2');
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
**children()**
closest(selector)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
**filterWhere(predicate)**
**find(selector)**
findWhere(predicate)
first()
forEach(fn)
get(index)
**hasClass(className)**
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue]
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
expect(
  wrapper
    .find('.todo-list')
    .children()
    .filterWhere(node => node.hasClass('completed'))
    .length()
).to.equal(3);
```

# Shallow Rendering

```
shallow(node[, options]) => ShallowWrapper
```

at(index)
childAt()
children()
closest(selector)
contains(nodeOrNodes)
context([key])
debug()
equals(node)
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
find(selector)
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialValue])
render()
setContext(context)
setProps(nextProps)
setState(nextState)
**shallow([options])**
**simulate(event[, data])**
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
describe('when the thumbnail is clicked', () => {
  const handleClick = sinon.spy();

  const wrapper = shallow(
    <Thumbnail path='foo.jpg' onClick={handleClick} />
  );

  wrapper.simulate('click');

  it('calls the click callback', () => {
    expect(handleClick.calledOnce).to.equal(true);
  });
})
```

# Full DOM Rendering - mount()

```
mount(node[, options]) => ReactWrapper
```

at(index)
contains(nodeOrNodes)
childAt()
children()
closest(selector)
context([key])
debug()
detach()
every(selector)
everyWhere(predicate)
filter(selector)
filterWhere(predicate)
find(selector)
findWhere(predicate)
first()
forEach(fn)
get(index)
hasClass(className)
html()
instance()
is(selector)
last()
map(fn)
**mount()**
not(selector)
parent()
parents()
prop([key])
props()
reduce(fn[, initialValue])
reduceRight(fn[, initialVa
ref(refName)
render()
setContext(context)
setProps(nextProps)
setState(nextState)
simulate(event[, data])
some(selector)
someWhere(predicate)
state([key])
text()
type()
unmount()
update()

```
import { mount } from 'enzyme';

describe('<Foo />', () => {

  it('calls componentDidMount', () => {
    spy(Foo.prototype, 'componentDidMount');
    const wrapper = mount(<Foo />);
    expect(Foo.prototype.componentDidMount.calledOnce).to.equal(true);
  });
```

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

# Examples

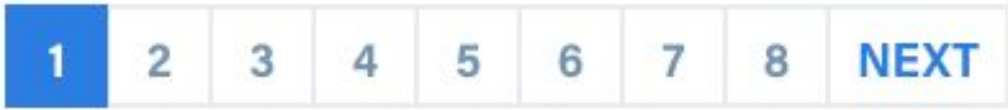

```
MediaGallery
  when there are no images
    ✓ does not render any images
  when there is one image
    ✓ renders one featured image
  when there are two images
    ✓ renders two featured images
  when there are three images
    ✓ renders the first image as featured and the rest normal
  when there are four images
    ✓ renders the first image as featured and the rest normal
  when there are five images
    ✓ renders the first image as featured and the rest normal
  when there are more than five images
    ✓ renders the first five images
    ✓ renders the first image as featured and the rest normal
    ✓ has a label with the full count of images on the last thumbnail
```

# Examples

```
<Rating />
  when the rating is 0
    ✓ renders five empty stars
    ✓ renders no half stars
    ✓ renders no full stars
  when the rating is 2
    ✓ renders two full stars
    ✓ renders no half filled stars
    ✓ renders three empty stars
  when the rating is 3.5
    ✓ renders three full stars
    ✓ renders one half filled star
    ✓ renders one empty star
  when the rating is 5
    ✓ renders no empty stars
    ✓ renders no half stars
    ✓ renders five full stars
```

# Examples

```
<PaginateButtons />
  previous button
    when the first page is active
      ✓ does not render a previous button
    when the first page is not active
      ✓ renders a previous button
  next button
    when the last page is active
      ✓ does not render a next button
    when the last page is not active
      ✓ renders a next button
  when a page is active
    ✓ is the only page with an active class applied
  on page change
    when the previous button is clicked
      ✓ calls onChange with the previous page number
    when the next button is clicked
      ✓ calls onChange with the next page number
    when a page button is clicked
      ✓ calls onChange with the page number
```



# Examples

```
110 passing (862ms)
```

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

- What is unit testing, benefits
- Automated javascript testing
- Unit testing redux
- Unit testing React components
- Real world examples
- Visual testing (Storybook)
- Resources

# Resources

Testing React Applications - April 2016 London React User Group

Test Driven React Tutorial - Spencer Dixon

Redux README - Writing Tests

Enzyme - Airbnb

Storybook - KadiraHQ

Slides: github.com/djkirby/react-symposium-unit-testing