

# Unit Testing React & Redux

Dylan Kirby

Slides: [github.com/djkirby/react-symposium-unit-testing](https://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

unit test

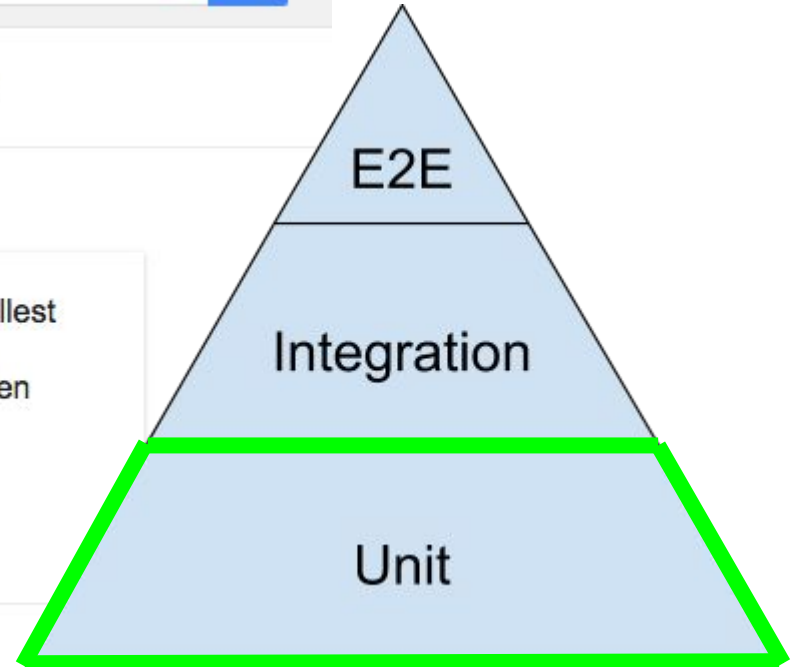
All Books Images Videos News More Search tools

About 30,700,000 results (0.53 seconds)

**Unit testing** is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. **Unit testing** is often automated but it can also be done manually.

[What is unit testing? - Definition from WhatIs.com](#)  
[searchsoftwarequality.techtarget.com/definition/unit-testing](https://searchsoftwarequality.techtarget.com/definition/unit-testing)

More about 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



<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

javascript test runner



All Shopping Images News Videos More Search tools

About 1,800,000 results (0.44 seconds)

### Karma - Spectacular Test Runner for Javascript

<https://karma-runner.github.io/>

Brings a productive testing environment to developers.

[How It Works](#) · [Installation](#) · [Configuration File](#) · [Configuration](#)

### GitHub - avajs/ava: Futuristic JavaScript test runner

<https://github.com/avajs/ava>

Even though JavaScript is single-threaded, IO in Node.js can happen in parallel due to its async nature. AVA takes advantage of this and runs your tests concurrently, which is especially beneficial for IO heavy tests. ... Switching from Mocha to AVA in Pageres brought the test time ...

### Wallaby - intelligent test runner for JavaScript, TypeScript, and ...

<https://wallabyjs.com/>

Smart and fast test runner for JavaScript/TypeScript/CoffeeScript, it enables continuous testing, TDD and BDD with instant feedback in your text editor.

### Mocha - the fun, simple, flexible JavaScript test framework

<https://mochajs.org/>

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making ..... which contains a Mocha test runner, among other things.

### Chutzpah - A JavaScript Test Runner - Home

<https://chutzpah.codeplex.com/>

Sep 16, 2014 - Open source test runner which runs unit tests using QUnit, Jasmine, Mocha, CoffeeScript and TypeScript.

```
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).toEqual(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`

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

```
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:

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

# Shallow Rendering

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

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

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

# Shallow Rendering

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

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

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



# Shallow Rendering

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

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

```
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`

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

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

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

```
  it('calls componentDidMount', () => {
```

```
    spy(Foo.prototype, 'componentDidMount');
```

```
    const wrapper = mount(<Foo />);
```

```
    expect(Foo.prototype.componentDidMount.calledOnce).toEqual(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

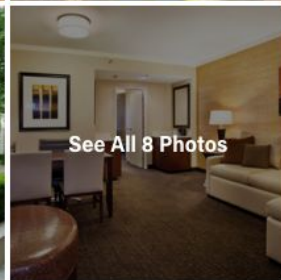
- ✓ renders the first image as featured and the rest normal

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

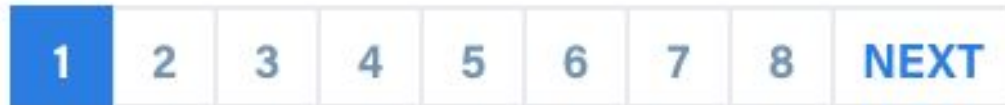
✓ calls `onChange` with the previous page number

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](https://github.com/djkirby/react-symposium-unit-testing)