

Testing React



Cory House

@housecor

reactjsconsulting.com



Here's the plan



Testing Technologies

- Testing frameworks
- Assertion libraries
- Helper libraries

We'll test

- React presentation components



Testing Frameworks





Test Frameworks

Mocha

Jasmine

Jest

Tape

AVA

Mocha

Serial

No assertions built in

Uses globals

No built-in ES6 support

Runs all tests on save

Long stack trace upon error

Proven, mature, with huge ecosystem

AVA

Concurrent

Assertions built in

No globals

Built-in ES6 support

Runs only impacted tests

Short specific error with marker

New





Test Frameworks



Cory House

@housecor

What JavaScript test framework should I use in my new "React and Redux in ES6" Pluralsight course?

59% Mocha

13% AVA

19% Tape

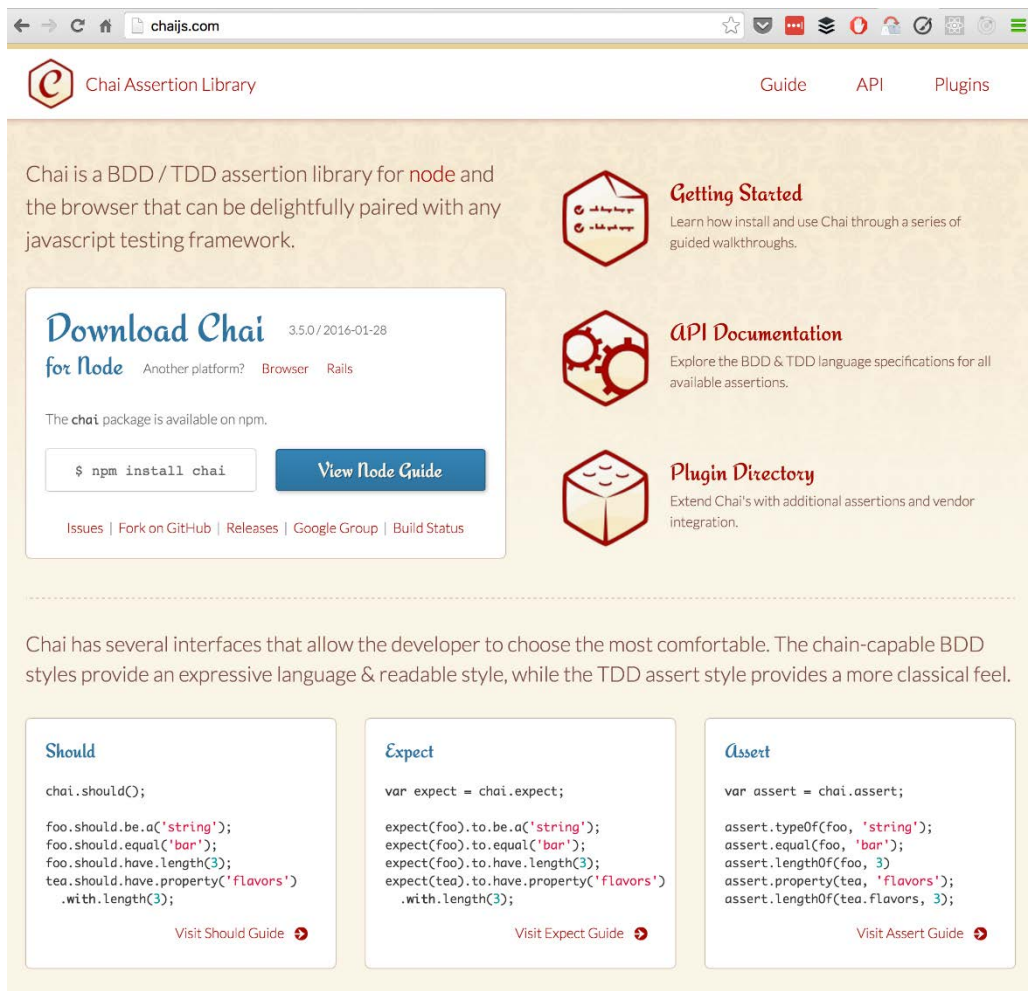
9% Jest

78 votes • Final results



Assertion Libraries

We'll use this



The screenshot shows the Chai Assertion Library website. The header includes the Chai logo and navigation links for Guide, API, and Plugins. The main content area features a 'Download Chai for Node' section with a version of 3.5.0 (2016-01-28) and a 'View Node Guide' button. Below this, there are three columns of links: 'Getting Started', 'API Documentation', and 'Plugin Directory'. At the bottom, there are three boxes for 'Should', 'Expect', and 'Assert' styles, each with a 'Visit [Style] Guide' link.

Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

Download Chai 3.5.0 / 2016-01-28
for Node Another platform? Browser Rails

The **chai** package is available on npm.

`$ npm install chai` [View Node Guide](#)

[Issues](#) | [Fork on GitHub](#) | [Releases](#) | [Google Group](#) | [Build Status](#)

Getting Started
Learn how install and use Chai through a series of guided walkthroughs.

API Documentation
Explore the BDD & TDD language specifications for all available assertions.

Plugin Directory
Extend Chai's with additional assertions and vendor integration.

Chai has several interfaces that allow the developer to choose the most comfortable. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel.

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);
```

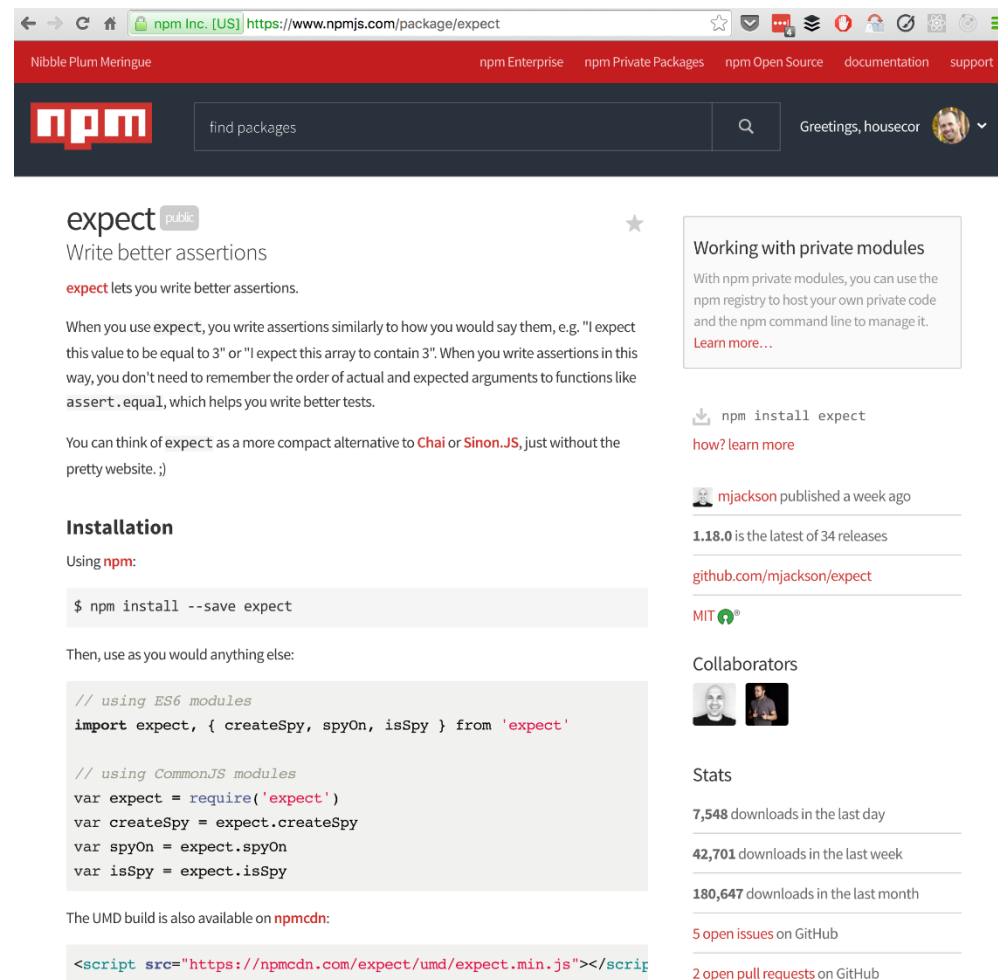
[Visit Expect Guide](#)

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](#)



The screenshot shows the npm package page for 'expect'. The header includes the npm logo and navigation links for npm Enterprise, npm Private Packages, npm Open Source, documentation, and support. The main content area features a 'Working with private modules' section, a 'Collaborators' section, and a 'Stats' section. The 'Installation' section shows the command to install expect and a code snippet for using expect in ES6 and CommonJS modules.

Nibble Plum Meringue npm Enterprise npm Private Packages npm Open Source documentation support

npm find packages [Greetings, housecor](#)

expect public
Write better assertions.

expect lets you write better assertions.

When you use expect, you write assertions similarly to how you would say them, e.g. "I expect this value to be equal to 3" or "I expect this array to contain 3". When you write assertions in this way, you don't need to remember the order of actual and expected arguments to functions like `assert.equal`, which helps you write better tests.

You can think of expect as a more compact alternative to **Chai** or **Sinon.JS**, just without the pretty website. ;)

Installation

Using **npm**:

```
$ npm install --save expect
```

Then, use as you would anything else:

```
// using ES6 modules
import expect, { createSpy, spyOn, isSpy } from 'expect'

// using CommonJS modules
var expect = require('expect')
var createSpy = expect.createSpy
var spyOn = expect.spyOn
var isSpy = expect.isSpy
```

The UMD build is also available on **npmcdn**:

```
<script src="https://npmcdn.com/expect/umd/expect.min.js"></script>
```

Working with private modules
With npm private modules, you can use the npm registry to host your own private code and the npm command line to manage it.
[Learn more...](#)

[npm install expect](#)
[how? learn more](#)

[mjackson](#) published a week ago
1.18.0 is the latest of 34 releases
[github.com/mjackson/expect](#)

MIT

Collaborators

Stats

7,548 downloads in the last day
42,701 downloads in the last week
180,647 downloads in the last month
5 open issues on GitHub
2 open pull requests on GitHub



★ expect public

Write better assertions

expect lets you write better assertions.

When you use **expect**, you write assertions similarly to how you would say them, e.g. "I expect this value to be equal to 3" or "I expect this array to contain 3". When you write assertions in this way, you don't need to remember the order of actual and expected arguments to functions like `assert.equal`, which helps you write better tests.

You can think of **expect** as a more compact alternative to **Chai** or **Sinon.JS**, just without the pretty website. ;)

Installation

Using **npm**:

```
$ npm install --save expect
```

Then, use as you would anything else:

```
// using ES6 modules
import expect, { createSpy, spyOn, isSpy } from 'expect'

// using CommonJS modules
var expect = require('expect')
var createSpy = expect.createSpy
var spyOn = expect.spyOn
var isSpy = expect.isSpy
```

The UMD build is also available on **npmcdn**:

```
<script src="https://npmcdn.com/expect/umd/expect.min.js"></script>
```

You can find the library on `window.expect`.

Working with private modules

With npm private modules, you can use the npm registry to host your own private code and the npm command line to manage it. [Learn more...](#)

📄 npm install expect

[how? learn more](#)

 **mjackson** published a week ago

1.18.0 is the latest of 34 releases

github.com/mjackson/expect

MIT 

Collaborators



Stats

7,548 downloads in the last day

42,701 downloads in the last week

180,647 downloads in the last month

5 open issues on GitHub

2 open pull requests on GitHub



Chai

to.equal

to.deep.equal

to.exist

to.not.exist

to.be.above

No spy

Expect

toBe

toEqual

toExist

toNotExist

ToBeGreaterThan

Spy built in



Helper Libraries



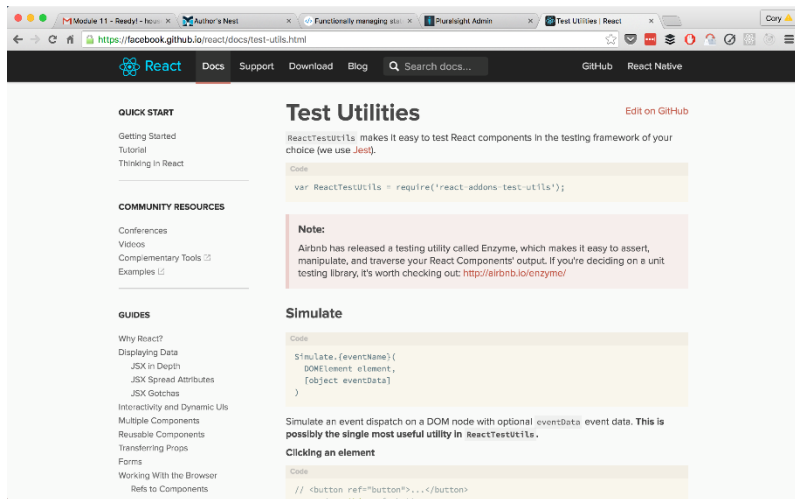


Helper Libraries

React Test Utils

Enzyme

React Test Utils



React testing library

Built by Facebook

Verbose API

React Test Utils: Two Rendering Options

shallowRender

Render single component

No DOM Required

Fast and Simple

renderIntoDocument

Render component and children

DOM Required

Supports simulating interactions



React Test Utils: DOM Interactions

- `findRenderedDOMComponentWithTag`
- `scryRenderedDOMComponentsWithTag`
- `Simulate`
 - Clicks
 - Keypresses
 - Etc.

Much more: facebook.github.io/react/docs/test-utils.html





Docs

Support

Download

Blog

Search docs...

GitHub

React Native

QUICK START

Getting Started

Tutorial

Thinking in React

COMMUNITY RESOURCES

Conferences

Videos

Complementary Tools

Examples

GUIDES

Why React?

Displaying Data

JSX in Depth

JSX Spread Attributes

JSX Gotchas

Interactivity and Dynamic UIs

Multiple Components

Reusable Components

Transferring Props

Forms

Working With the Browser

Refs to Components

Test Utilities

[Edit on GitHub](#)

`ReactTestUtils` makes it easy to test React components in the testing framework of your choice (we use `Jest`).

Code

```
var ReactTestUtils = require('react-addons-test-utils');
```

Note:

Airbnb has released a testing utility called Enzyme, which makes it easy to assert, manipulate, and traverse your React Components' output. If you're deciding on a unit testing library, it's worth checking out: <http://airbnb.io/enzyme/>

Simulate

Code

```
Simulate.{eventName}(  
  DOMElement element,  
  [object eventData]  
)
```

Simulate an event dispatch on a DOM node with optional `eventData` event data. **This is possibly the single most useful utility in `ReactTestUtils`.**

Clicking an element

Code

```
// <button ref="button">...</button>
```



Why Enzyme?

React Test Utils

`findRenderedDOMComponentWithTag`

`scryRenderedDOMComponentsWithTag`

`scryRenderedDOMComponentsWithClass`

Enzyme

`find`

`find`

`find`



Enzyme Is an Abstraction



Behind the scenes

- React Test Utils
- JSDOM (In-memory DOM)
- Cheerio (Fast jQuery style selectors)

Comparison

	React Test Utils	JSDOM	Cheerio	Enzyme
Run tests in	Node	Node	Node	Node
Dependencies	None	None	None	JSDOM, Cheerio, React Test Utils
Selectors	Wordy	DOM	jQuery	Terse
Interactions	Yes	Yes	No	Yes
Ease	Clunky	Okay	Easy	Easy
Speed	Varies	Okay	Fast	Varies



Where to Test



Where to Test

```
Author Actions
  ✓ should create CREATE_AUTHOR_SUCCESS action

Async Actions
  ✓ should create LOAD_AUTHORS_SUCCESS when authors have been loaded (1006ms)

Course Actions
  ✓ should create a END_CREATE_COURSE action

Async Actions
  ✓ should create END_LOAD_COURSES when courses have been loaded (1006ms)

AJAX Call Status Reducer
  ✓ should increment the number of calls in progress
  ✓ should decrement the number of calls in progress when any action ending in
  ✓ should decrement the number of calls in progress when API_CALL_ERROR is di

Author Reducer
  ✓ should add author
  ✓ should create a new object when creating a new author
  ✓ should remove author
  ✓ should update author

Course Reducer
  ✓ should add course
  ✓ should create a new object when creating a new course
  ✓ should remove course
  ✓ should update course
```

Browser

Headless Browser

In-memory DOM



Naming Test Files



Cory House

@housecor

How do you prefer to name your **#JavaScript** test files?

46% fileName.spec.js

39% fileName.test.js

15% Other - Please reply

180 votes • 2 hours left



Where Do Test Files Belong?

Mocha default: /test

We'll place tests alongside the file under test. Why?

- Easy imports. Always `./filenameUnderTest`
- Clear visibility
- Convenient to open
- Move files and tests together



Our Plan



What

React components and Redux

How

Mocha with Expect

Where

In-memory DOM via JSDOM

Helper

Enzyme

Testing Presentation Components



Rendering Options

Shallow Render

Fast

Test in isolation

No refs or interactions yet

Recommended for the future

Render into Document

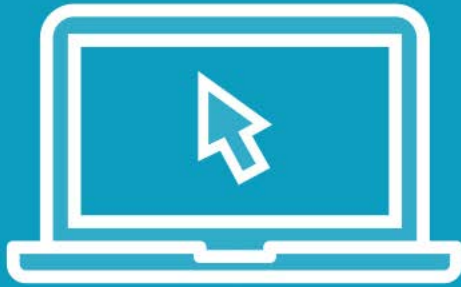
Slower

Test set of components

Test refs and interactions



Demo

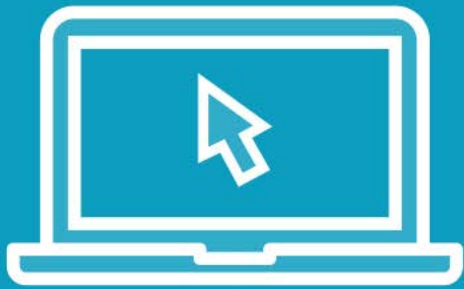


Test Component with Shallow Rendering

- React Test Utils
- Enzyme



Demo



Enzyme



Wrap up



Testing Approach

- Mocha, Expect, React Test Utils, and Enzyme
- In-memory DOM via Node

Tested

- React presentation components

Next up: Testing Redux

