

# Testing and Continuous Integration

---



**Cory House**

@housecor

bitnative.com



# Here's the Plan



**JavaScript testing styles**

**6 key testing decisions**

**Configure testing and write test**

**Continuous integration**



# JavaScript Testing Styles



Style	Focus
Unit	Single function or module
Integration	Interactions between modules
UI	Automate interactions with UI



# Unit Testing Decisions

1

Framework

2

Assertion Library

3

Helper Libraries

4

Where to run tests

5

Where to place tests

6

When to run tests



# Decision #1: Testing Framework



# Testing Frameworks



Mocha



Jasmine



Tape



QUnit



AVA



Jest

# It's Like Choosing a Gym



The important part is showing up.



## Right Answer



Any of these!

## Wrong Answer

Woah, decision fatigue!

I'll just keep coding  
and praying.





# Decision #2: Assertion Library



# What's An Assertion?

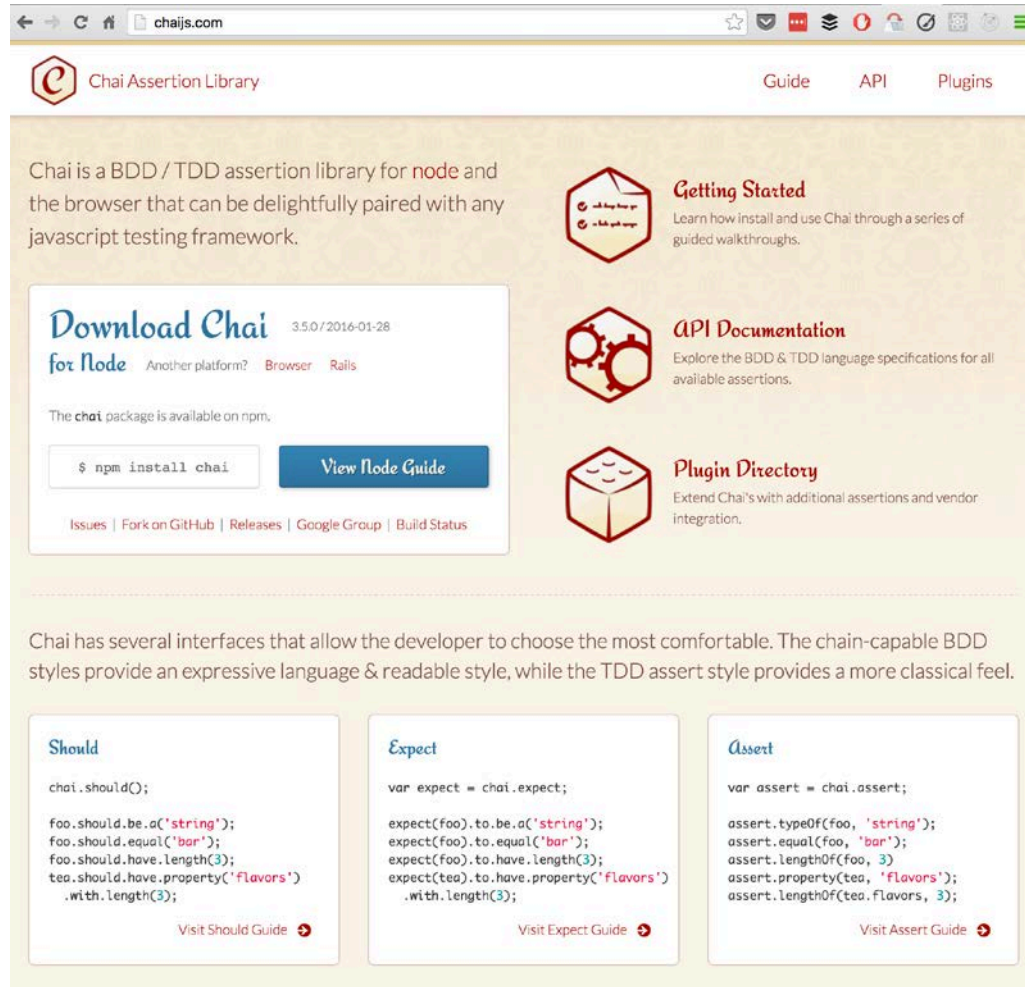


**Declare what you expect**

```
expect(2+2).toEqual(4)
```

```
assert(2+2).equals(4)
```

# Assertion Libraries



The screenshot shows the Chai Assertion Library website. At the top, there's a navigation bar with 'Guide', 'API', and 'Plugins'. The main content area has a large heading 'Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.' Below this, there's a 'Download Chai for Node' section with a version '3.5.0/2016-01-28' and a button to 'View Node Guide'. To the right, there are three sections: '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](#)

## Should.js

### global

```
Assertion.add
Assertion.addChain
Assertion.alias
PromisedAssertion
should
should.Assertion
should.AssertionError
should.config
should.extend
should.noConflict
should.use
```

### assertion

```
Assertion#any
Assertion#assert
Assertion#fail
Assertion#not
```

### assertion assert

```
should.deepEqual
should.doesNotThrow
should.equal
```

## "global" Members

### Assertion.add(name, func)



Way to extend Assertion function. It uses some logic to define only positive assertions and itself rule with negative assertion. All actions happen in subcontext and this method take care about negation. Potentially we can add some more modifiers that does not depends from state of assertion.

#### Arguments

1. name (*String*): Name of assertion. It will be used for defining method or getter on Assertion.prototype
2. func (*Function*): Function that will be called on executing assertion

#### Example

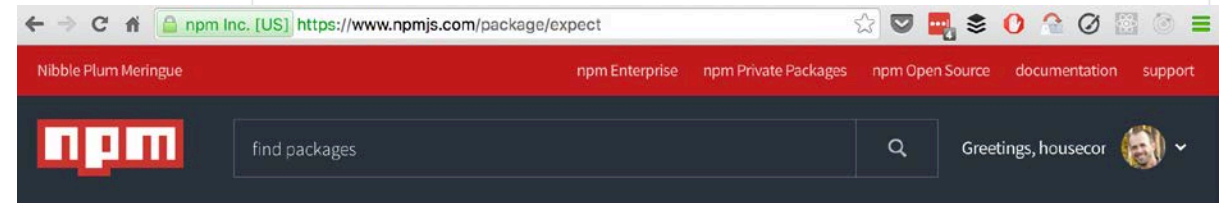
```
Assertion.add('asset', function() {
  this.params = { operator: 'to be asset' }

  this.obj.should.have.property('id').which.is.a.Number()
  this.obj.should.have.property('path')
})
```

### Assertion.addChain(name, [onCall])



Add chaining getter to Assertion like .a, .which etc



The screenshot shows the npm website. At the top, there's a navigation bar with 'npm Enterprise', 'npm Private Packages', 'npm Open Source', 'documentation', and 'support'. Below this, there's a search bar with the text 'find packages' and a search icon. To the right, there's a user profile section with the text 'Greetings, housecor' and a user avatar. The main content area is currently empty.

## 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:

### 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](https://github.com/mjackson/expect)

MIT

Collaborators

# Decision #3: Helper Library



# JSDOM

**Simulate the browser's DOM**

**Run DOM-related tests without a browser**



# Cheerio

**jQuery for the server**

**Query virtual DOM using jQuery selectors**



# Decision #4: Where to Run Tests



# Where to Run Tests

```
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

- Karma, Testem

## Headless Browser

- PhantomJS

## In-memory DOM

- JSDOM





Decision #5:  
Where do test files belong?



# Where Do Test Files Belong?

## Centralized

Less “noise” in src folder

Deployment confusion

Inertia

## Alongside

Easy imports

Clear visibility

Convenient to open

No recreating folder structure

Easy file moves

Path to file under test is always `./filename` 😊

```
// file.test.js
import file from '../../../../src/long/path'
```

```
// file.test.js
import file from './file'
```



# 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



Decision #6:  
When should tests run?



# Unit Tests Should Run When You Hit Save



**Rapid feedback**

**Facilitates TDD**

**Automatic = Low friction**

**Increases test visibility**



But Cory, my tests  
are too slow!

- You, my viewer with slow tests



## Unit Tests

Test a small unit

Often single function

Fast

Run upon save

## Integration Tests

Test multiple units

Often involves clicking and waiting

Slow

Often run on demand, or in QA



# Here's the Plan

1

Framework  
**Mocha**

2

Assertion Library  
**Chai**

3

Helper Libraries  
**JSDOM**

4

Where to run tests  
**Node**

5

Where to place tests  
**Alongside**

6

When to run tests  
**Upon save**





# Demo



## Configure automated testing



# Continuous Integration

---





Weird.

Works on my  
machine.



# Why CI?



Forgot to commit new file

Forgot to update package.json

Commit doesn't run cross-platform

Node version conflicts

Bad merge

Didn't run tests

Catch mistakes quickly



# What Does a CI Server Do?



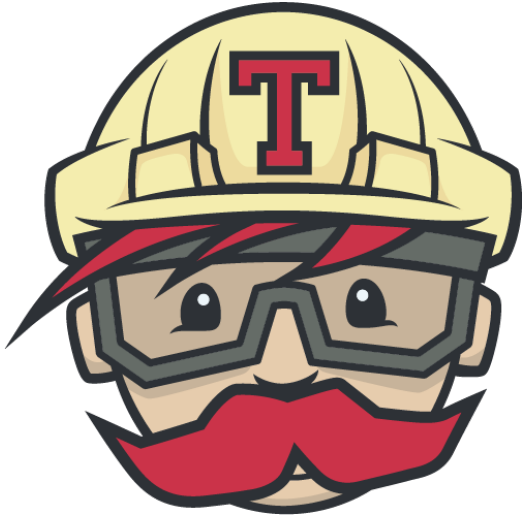
**Run Automated build**

**Run your tests**

**Check code coverage**

**Automate deployment**

# Continuous Integration



Travis



Appveyor



Jenkins

# Demo



## Set up Continuous Integration



## Wrap Up



### Testing frameworks

- Mocha, Jasmine, AVA, Tape, Jest..

### Assertion libraries

- Chai, Expect

### Helper libraries

- JSDOM, Cheerio

### Where to run tests

- Browser, Headless, In memory

### Where to place tests, and when to run

### Continuous Integration

- Travis CI, Appveyor, Jenkins

### Next up: HTTP and Mock APIs

