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

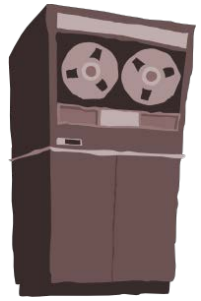
When to run tests

Decision #1:
Testing Framework

Testing Frameworks

Mocha

Jasmine



Tape

QUnit

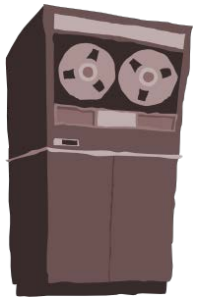
AVA

Jest

It’s Like Choosing a Gym

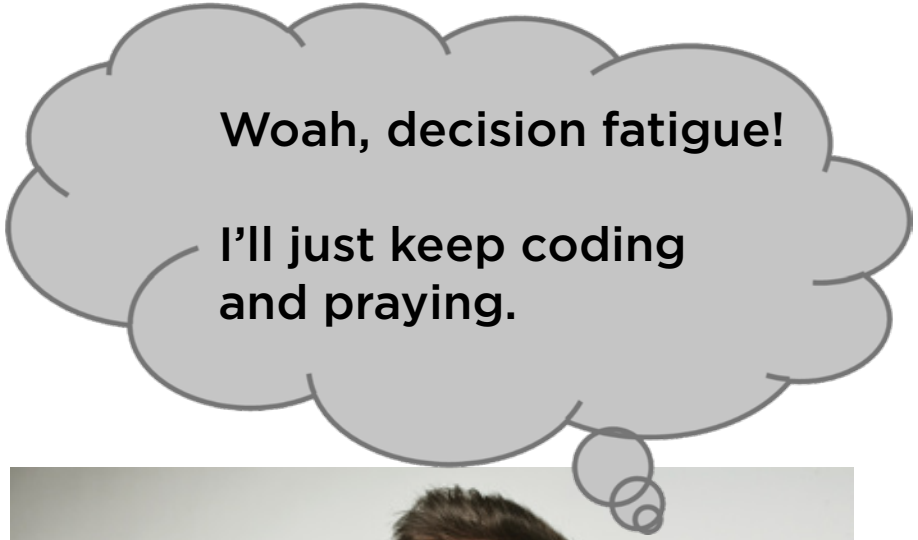
The important part is showing up.

Right Answer



Any of these!

Wrong Answer



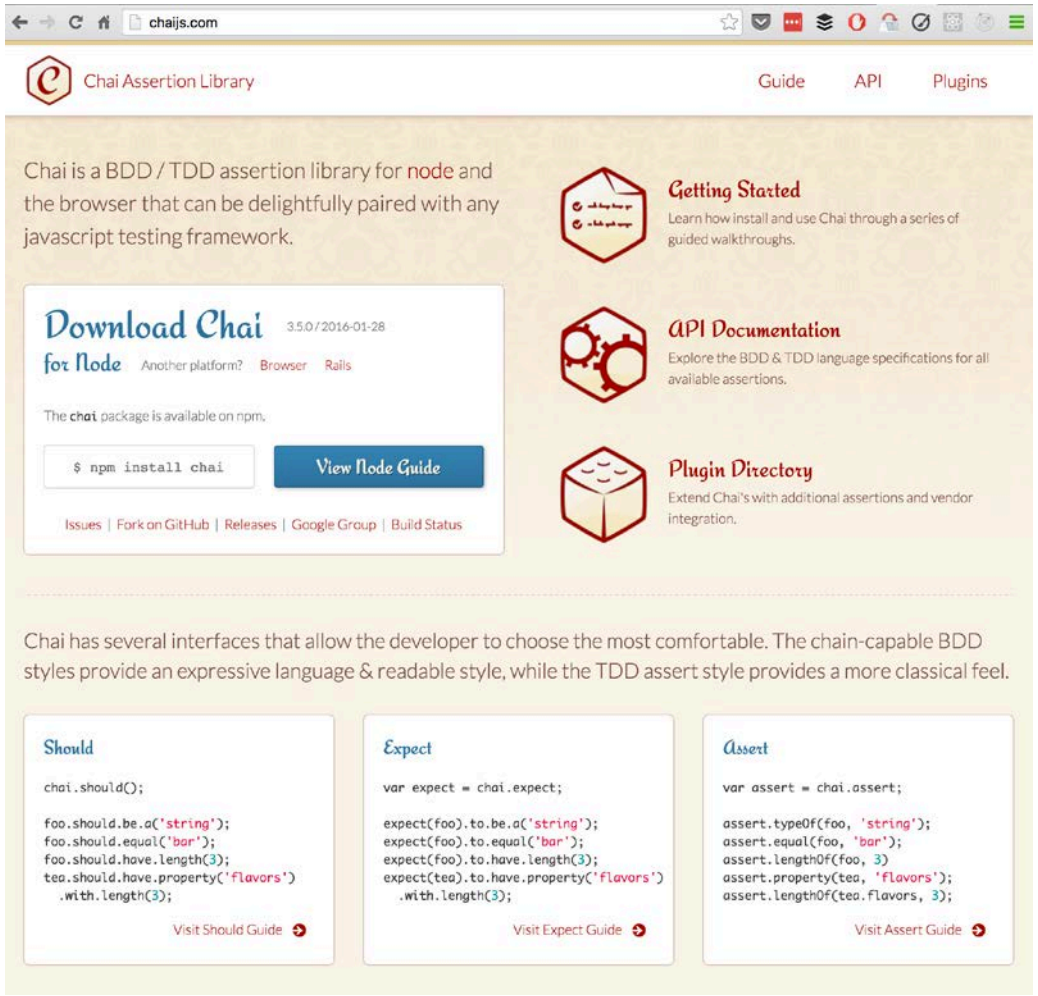
Decision #2:
Assertion Library

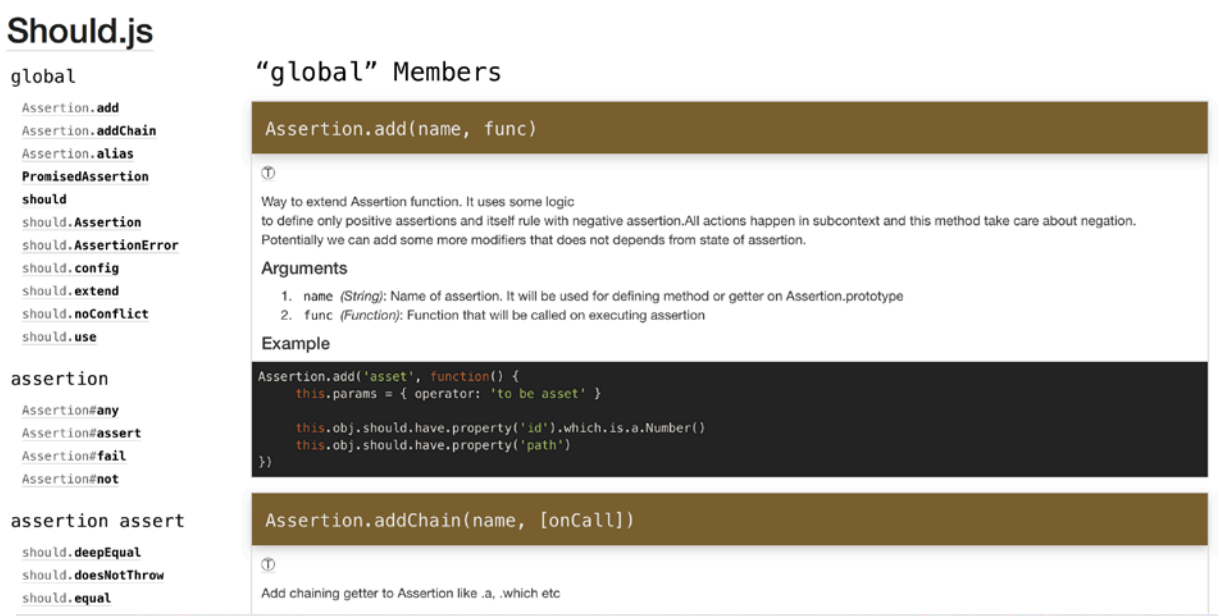
What's An Assertion?

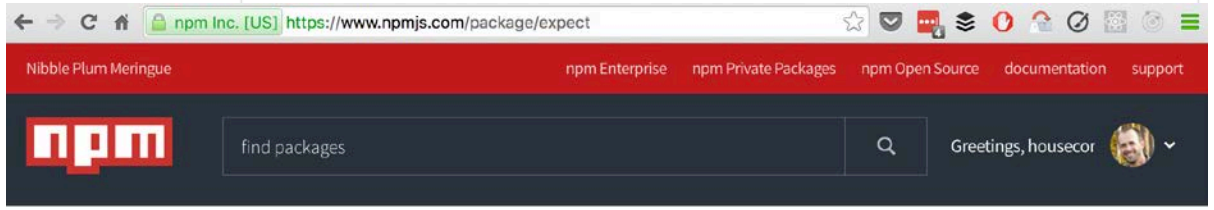
Declare what you expect

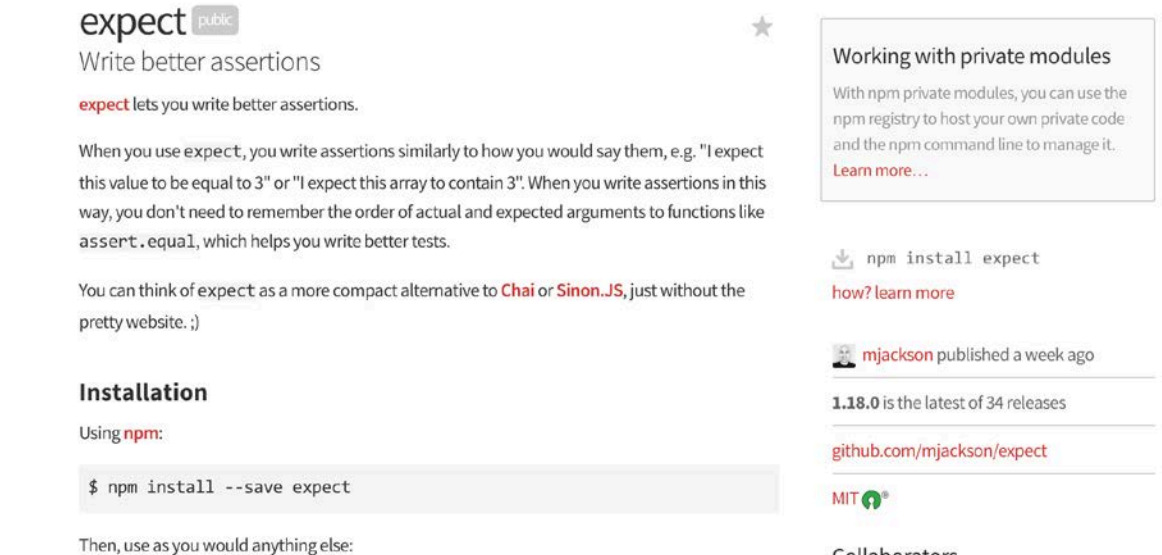
```
expect(2+2).to.equal(4)

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

Assertion Libraries









Decision #3:
Helper Library

JSDOM

Simulate the browser's DOM

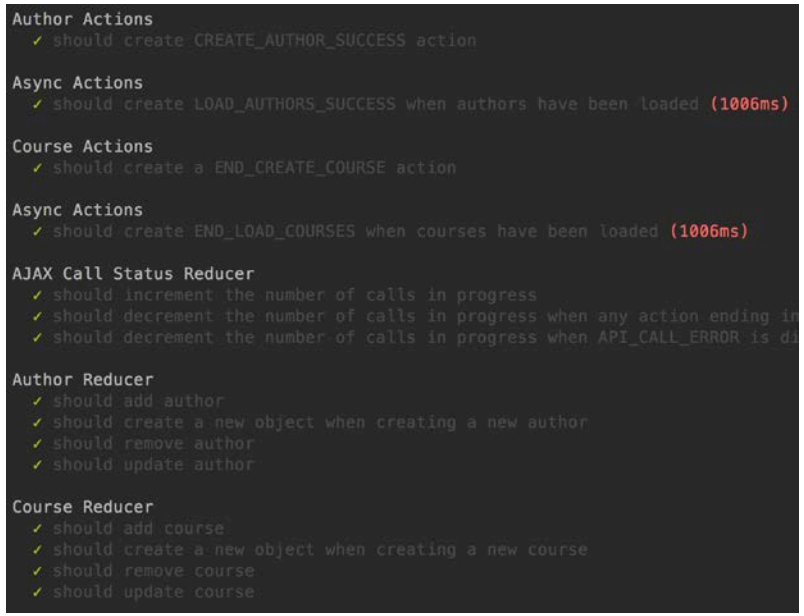
Run DOM-related tests without a browser

Cheerio

jQuery for the server

Query virtual DOM using jQuery selectors

Decision #4:
Where to Run Tests



Where to Run Tests

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

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

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 './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	Integration Tests
Test a small unit	Test multiple units
Often single function	Often involves clicking and waiting
Fast	Slow
Run upon save	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

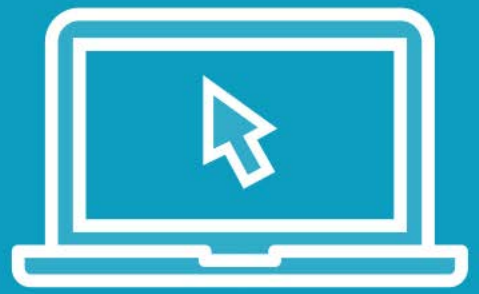
5

Where to place tests
Alongside

6

When to run tests
Upon save

Demo



Configure automated testing

Continuous Integration

Weird.

Works on my machine.

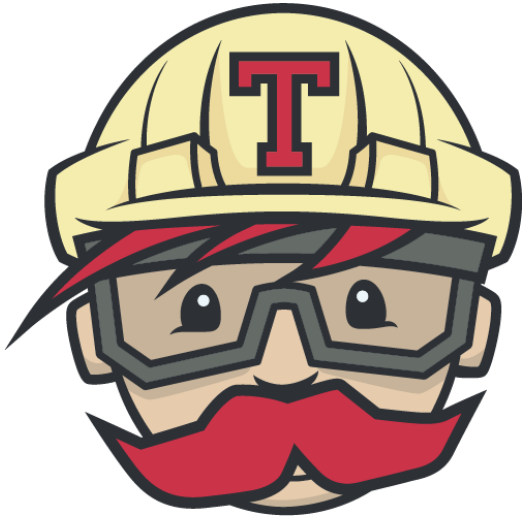
Why CI?

- 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

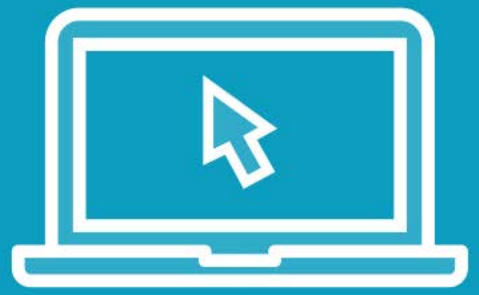


Travis

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