Testing and Continuous Integration

Cory House

@housecor reactjsconsulting.com

Building a JavaScript Development Environment

by Cory House

Starting a new JavaScript project from scratch is overwhelming. This course provides a playbook outlining the key decisions you need to make. Build a robust development environment that handles bundling, linting, transpiling, testing, and much more.

Resume Course | Bookmark | Add to Channel | Live mentoring

Table of contents | Description | Transcript | Exercise files | Discussion | Recommended

This course is part of: Node.js Path

Expand all

Course Overview — 1m 30s

Course author

Cory House

Cory is an independent consultant with over 15 years of experience in software development. He is a Microsoft MVP, ASP Insider, and a member of the Telerik developer experts program.

Course info

Level	Beginner
Rating	★★★★★ (209)
My rating	★★★★★
Duration	5h 19m
Released	9 Nov 2016

Share course

What do you want to learn?

Cory
housecor@gmail.com

The Plan

JavaScript testing styles

6 key testing decisions

Write tests

Continuous integration

Automated Testing Decisions

1

Framework

2

Assertion Library

3

Helper Libraries

4

Where to run tests

5

Where to place tests

6

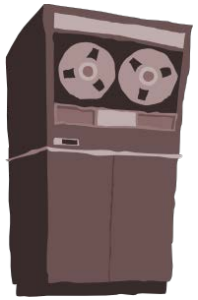
When to run tests

Decision:
Testing Framework

Testing Frameworks

Mocha

Jasmine



Tape

QUnit

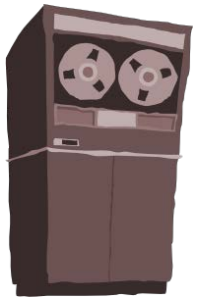
AVA

Jest

It’s Like Choosing a Gym

The important part is showing up.

Right Answer



Any of these!

Wrong Answer

- Fast
- Only runs tests related to changed code
- Interactive watch mode
- Snapshot testing
- Helpful error messages
- Debug via console or breakpoints
- Mocks and spies
- Coverage reporting
- Simple configuration
- Built into create-react-app

Decision:
Types of Tests

gist.github.com/thevangelist/e2002bc6b9834def92d46e4d92f15874

Create a new ta

GitHub Gist Search... All gists GitHub New gist

thevangelist / my-component.spec.js

Created 6 months ago

Star 102 Fork 8

Code Revisions 1 Stars 102 Forks 8

Embed <script src="https://gis Download ZIP

The only React.js component test you'll ever need (Enzyme + Chai)

my-component.spec.js

Raw

```
import React from 'react';
import { shallow } from 'enzyme';
import MyComponent from '../src/my-component';

const wrapper = shallow(<MyComponent/>);

describe('(Component) MyComponent', () => {
  it('renders without exploding', () => {
    expect(wrapper).to.have.length(1);
  });
});
```

UI Testing Types

Unit

Interaction

Structural

Style

Unit Testing

Goal: Validate logic

- Validation
- Calculations
- Transformations
- We'll use Jest

When to Unit Test

1. Business logic
2. Individual modules/functions

Interaction Testing

Goal: Validate interactivity

- Confirm interactions work
- Assert a function is called on click
- Message displays upon submission
- No browser required!
- We'll use Enzyme

When to Interaction Test

1. Interactive component
2. Validate events
3. Non-deterministic markup

Structural Testing

Goal: Validate HTML output

- Jest offers snapshot testing
- Save copy of output for a given input
- Tests break when output changes
- The only test needed if stateless

When to Snapshot Test

1. Non-interactive component
2. Deterministic markup
3. Validate display/hiding
4. Test display after interactions

Why Snapshot Testing?

1. Easy set up
2. Notified when rendering changes
3. Update with a single key
4. Fast - No browser. In-memory.
5. Debug like code

Snapshots != TDD

Write snapshot tests *after* development

Useful, but write other types of tests too!

Style Testing

Goal: Automated visual regression testing

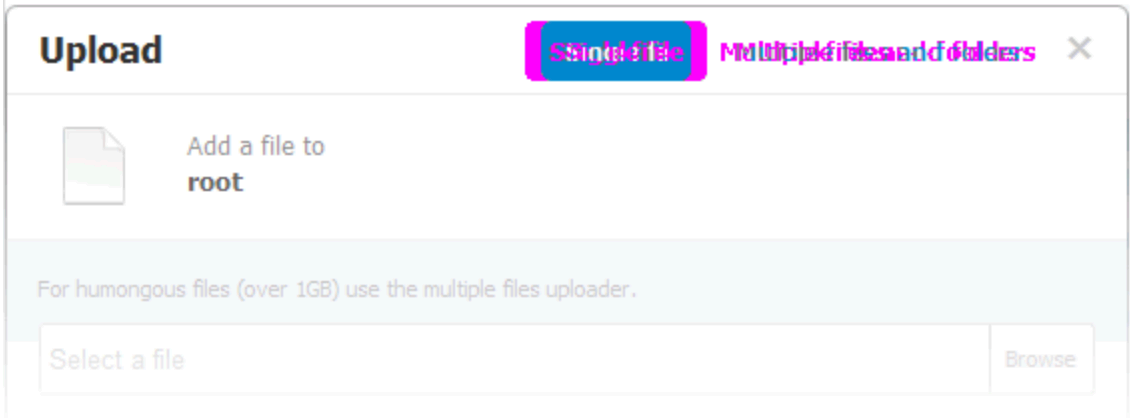
- Compare literal screenshots
- BackstopJS, PhantomCSS, Casper

PhantomCSS

CSS regression testing. A CasperJS module for automating visual regression testing with PhantomJS 2 or SlimerJS and Resemble.js. For testing Web apps, live style guides and responsive layouts. Read more on Huddle's Engineering blog: CSS Regression Testing.

What?

PhantomCSS takes screenshots captured by CasperJS and compares them to baseline images using Resemble.js to test for rgb pixel differences. PhantomCSS then generates image diffs to help you find the cause.



Style Testing

Goal: Automated visual regression testing

- Compare literal screenshots
- Arguably redundant with inline styles
- Uses actual browsers
- BackstopJS, PhantomCSS, Casper

Decision:
Assertion Library

What's an Assertion?

Declare what you expect

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

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

Decision:
Helper Libraries

Helper Libraries

Enzyme

- Interaction tests
- Simulate clicks
- DOM queries

react-test-renderer

- Render Jest Snapshots

Decision:
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
```

Where to Run Tests

Browser

- Karma, Testem

Headless Browser

- PhantomJS

In-memory DOM

- JSDOM

Decision:
Where do test files belong?

Where Do Test Files Belong?

Centralized	Alongside
Less “noise” in src folder	Easy imports
	Clear visibility
	Convenient to open
	No recreating folder structure
	Easy file moves

Decision:
When should tests run?

Run Tests When You Hit Save

Rapid feedback

Facilitates TDD

Automatic = Low friction

Here's the Plan

1

Framework

Jest

2

Assertion Library

Jest

3

Helper Libraries

Enzyme

4

Where to run tests

Node

5

Where to place tests

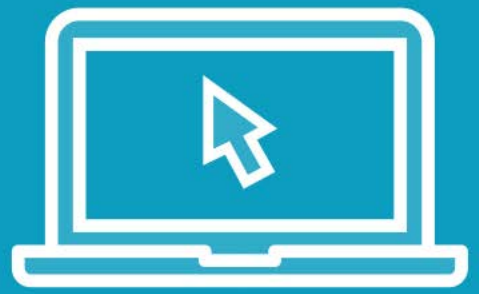
Alongside

6

When to run tests

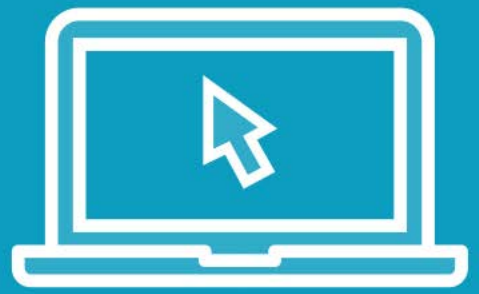
Upon save

Demo



Automated unit test

Demo

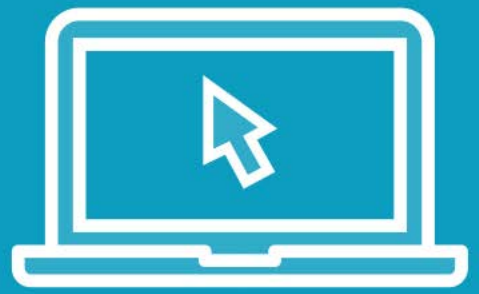


Snapshot test

Enzyme: Shallow vs Mount

- **Begin with shallow**
- **Need to test lifecycle methods or children? Use mount.**

Demo



Automated interaction test using Enzyme

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

circleci

CircleCI

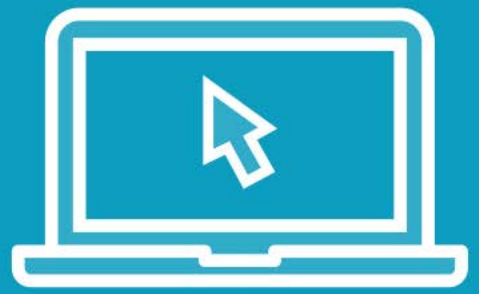
semaphore

Semaphore

snap

SnapCI

Demo



Set up continuous integration

Wrap Up

Testing frameworks

- Jest

Assertion libraries

- Built into Jest

Helper libraries

- Enzyme

Where to run tests

- Node with JSDOM

Place tests alongside, run upon save

Continuous Integration

- Travis CI

Next up: Distribution decisions