Fundamentals of Jest Testing

Daniel Stern
CODE WHISPERER

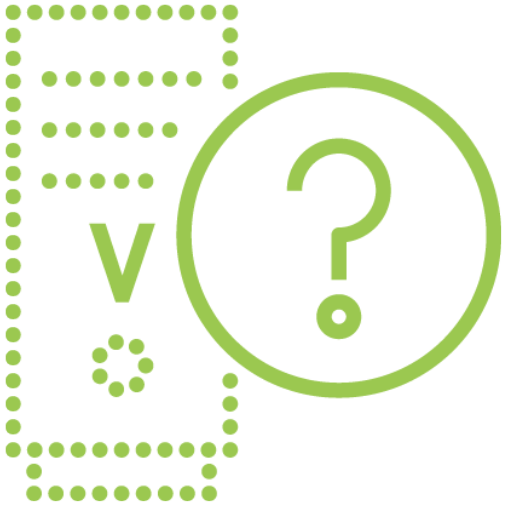
@danieljackstern

“It will probably work.”

– Everyone, at one point or another

Jest Installation

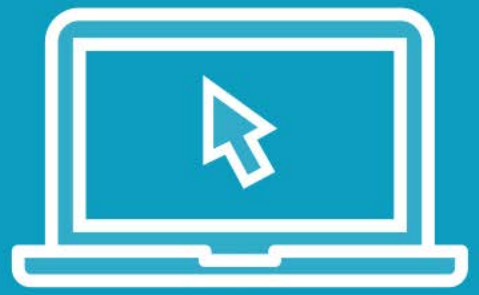
Installed via NPM like many other libraries



Local installation should determine version, but in practice CLI may call local or global installation

CI installs Jest and CLI automatically, usually based on package.json

Demo



Add local Jest installation to *Isomorphic React* application

Install Jest CLI globally, note the difference between global and local

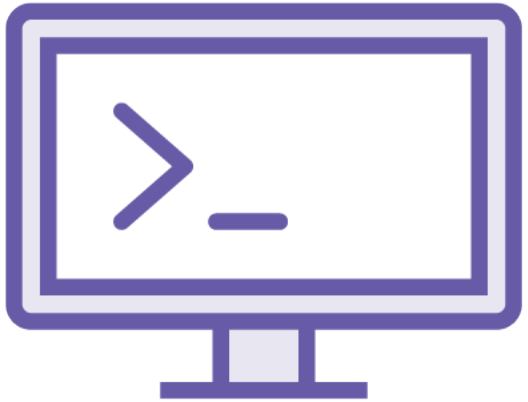
Confirm Jest CLI and local Jest installation are interacting as expected

Running Tests

“Well done is better than well said.”

– **Benjamin Franklin**

Running Tests



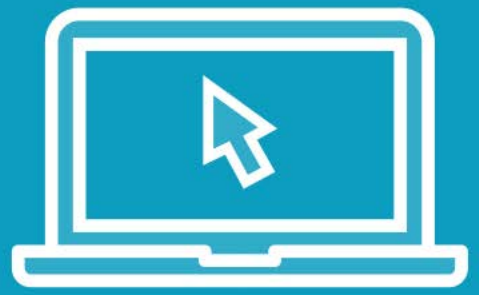
Tests are run by using the Jest CLI (typing “jest” followed by arguments in the command line)

test
test-watch
test-e2e
test-update
test-prod

The correct configuration for various different test patterns are stored as NPM scripts

In practice, tests are “run” by CI software and “watched” by everything else

Demo



Use Jest to run tests manually through the command line

Configure package.json with shortcuts for running Jest

Note that thread completes after test execution

Creating Test Files

How Are Test Files Identified?

__tests__/*.js

*.spec.js
*.test.js

Any files inside a folder named __*tests*__ are considered tests

Any files with *.spec* or *.test* in their filename are considered tests

Should Tests Be Alongside Components or in Their Own Folder?

Tests in Their Own Folder

Easily distinguish between test and non-test files

Unrelated files can share a folder (i.e., a “loginService” test and a “cryptoHash” test)

Very easy to isolate a particular set of tests that are in the same folder

Tests can be named anything but must be inside an appropriately named folder (i.e., *__tests__*), to be recognized

Tests Alongside Components

Which files are components and which files are spec is not as obvious

Tests are always adjacent to the files they apply to

Unrelated tests are less likely to share a folder

Tests must have the correct naming pattern to be recognized, i.e., **.test.js*

Possible to isolate tests based on name patterns, i.e., *user-**

Jest Globals

Describe and It

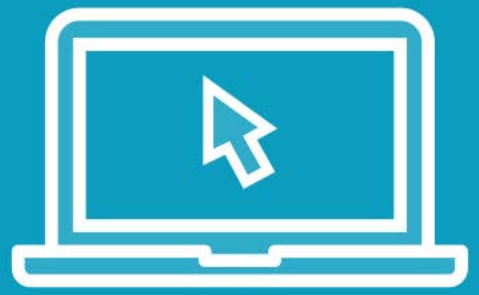
IT (TEST)

Method which you pass a function to, that function is executed as block of tests by the test runner

DESCRIBE (SUITE)

An optional method for grouping any number of *it* or *test* statements

Demo



Create a test file in a *__tests__* directory

Create a test file with *.spec*
as part of the filename

Use *describe* and *it* blocks

Verify that Jest automatically recognizes these test files

Test content will be added later

Watching for Changes

Watching Tests

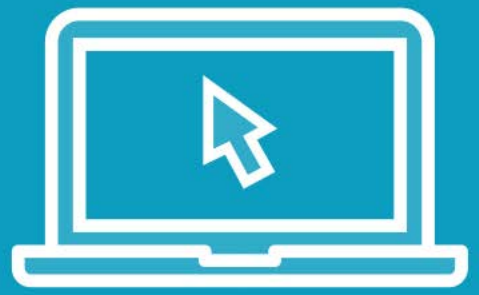
In “watch mode”, tests are run automatically as files change

Only tests pertaining to changed files are run

Jest detects changes automatically

Actively prevents regression

Demo



Initialize Jest watch via command line

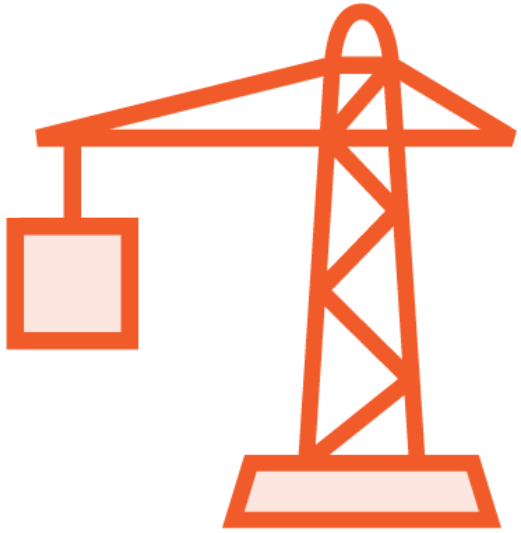
Note that tests are run again each time files are updated

Note that changed files are detected automatically by Jest

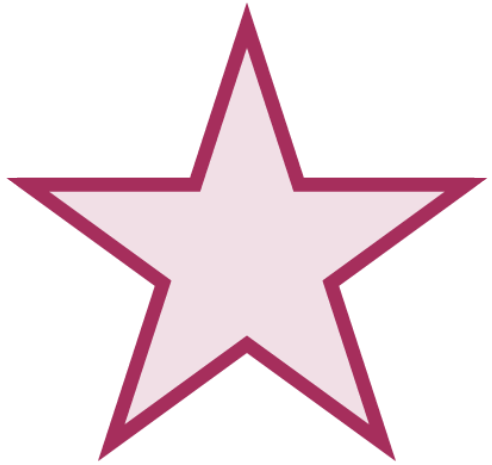
Setup and Teardown Globals

BeforeEach and BeforeAll

BeforeEach runs a block of code before each test



Useful for setting up databases, mock instances, etc.



BeforeAll runs code just once, before the first test

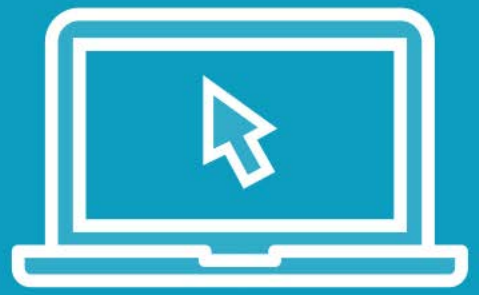
AfterEach and AfterAll

Inverse versions of BeforeEach and BeforeAll

Runs a block of code after each test (or after the last test)

Useful for closing open connections, terminating sub-processes

Demo

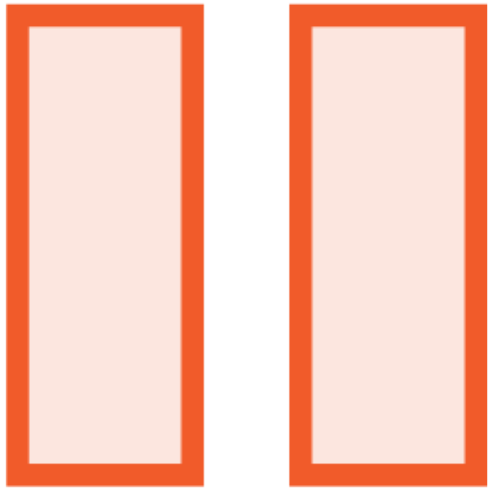


Add *before* and *after* blocks to an existing test

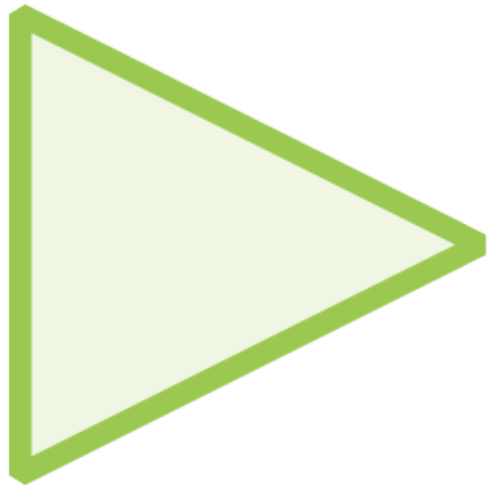
Note that blocks are executed automatically before and after test

We will use *beforeAll* to facilitate mocking (upcoming module)

Skipping and Isolating Tests

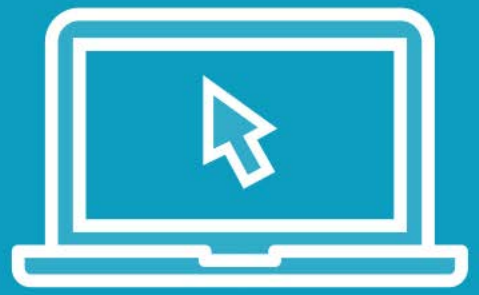


Skipping a test results in that test not being run



Isolating a test results in only it (and any other isolated tests) running

Demo



Mark a test as *skip* and note the change in suite of tests

Mark a test as *only* and note the corresponding change

Asynchronous Testing

What Are Asynchronous Tests?

Contains assertions (like a regular test)

Does not complete instantaneously

Can take varying amounts of time, even an unknown amount of time

Jest must be notified that test is complete

Defining Asynchronous Tests

Invoke the *done()* callback that is passed to the test

Return a promise from a test

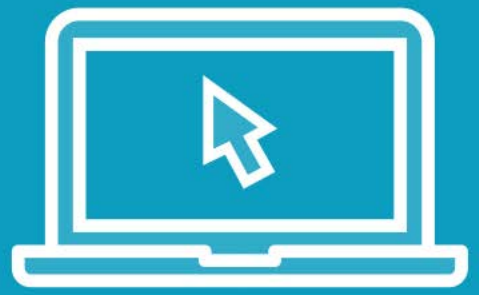
Pass an async function to *describe*

```
it("async test 1",done=>{
    setTimeout(done,100);
});

it("async test 2",()=>{
  return new Promise (
    resolve=>setTimeout(resolve,100)
  )
});

it("async test 3",
    async ()=>await delay(100)
);
```

- The ways of formatting an async test shown here are roughly equivalent

- Delay is a method that returns a promise

Demo



Write an asynchronous test with a callback

- Note that test hangs if callback is not called

Write a test with a promise

Write a test using async / await

Summary

Jest is installed, and tests are run and watched via the command line

Running tests is most useful for CI suites, watching tests is most useful for devs

Tests can either be in a *__tests__* folder or a have a *.spec* extension

Global configuration can be handled with *before* and *after* blocks

It is possible to skip a test, or run it in isolation

Jest supports a variety of asynchronous testing techniques

Coming up in the Next Module...

- What Are Mocks?
- The Require Global
- Mocking Modules
- The Spy Game
- Automatic and Manual Mocking