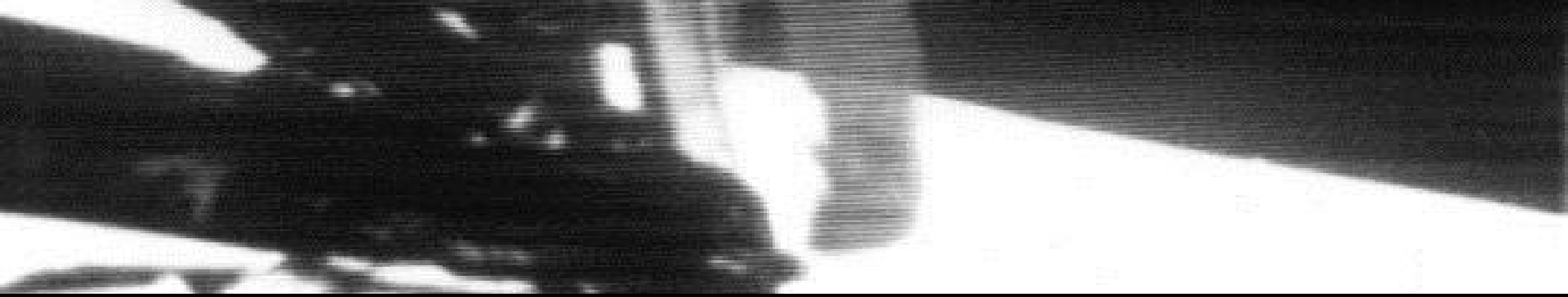

First Steps
Unit Testing JS

—

Open Source JS | 07.05.17 | Conny Kawohl

What is Unit Testing?

It is a software testing method by which individual units of source code, sets of one or more modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use

Phew. That sounds hard.


A daunting task?

What do we want to accomplish?

- Make sure we can test everything that is critical to the functionality of our app
- Get all the bugs out!
- Make it easier for other developers to work with our app
- Approximate existing user stories and subtasks as starting point for the tests
- So that non-developers can understand what’s happening
- Include a minimum of dependencies
- OPTIONAL: Integrate into existing task runner/build processes


It ain’t rocket science.

Behavior Driven Development

- Tests are written as whole sentences
- Using a Domain-specific language:
  - Starting with a description (“describe”),
  - Followed by a unit identifier (“it”)
  - And a conditional verb (“should”)
  - In order of business value.

- Thereby a whole sentence emerges when the test runs, giving non-developers the chance to understand what is being tested and why.
  - Example: “[Describe] Running the engine [it] *should* propel the rocket”


Unit tests save
~~lives~~ *nerves*

Why should I unit-test?

1. **Prevent bugs, save your nerves**
   Find them ahead of time and be on the safe side

2. **Understand your code better**
   Find out what your function *really* does!

3. **Think ahead, BDD-style**
   Write your tests before you code, describing your intent before finding a suitable solution.

4. **Automate testing**
   ...and let the computer do your work!


Getting there
mocha

Getting there: Install Mocha

1. **Install Mocha in your project**

   ```
   $ npm i mocha --save-dev
   ```

2. **Create a “test” subdirectory**

   ```
   $ mkdir test
   ```

3. **Change into the test subdirectory**

   ```
   $ cd test
   ```

4. Create an empty test file

   ```
   $ touch app_test.js
   ```

   Naming convention: Use the name of the primary JS file, add “test”, separated by underscore or dot.

Getting there: Write first assertion

1. **Open the test file in your favorite editor and write your first test.**

```
var assert = require('assert');

describe('Igniting the jet fuel', function() {
    it('should start the Engine', function() {
        assert.ok(true);
    });
});
```


A helping hand
chai

A helping hand: Install Chai

1. **Install Mocha in your project**

   ```
   $ npm i chai --save-dev
   ```

2. **Require chai in the second line of your app_test.js**

   ```
   var assert = require('chai').assert
   ```

Getting there: Assert with Chai

1. **Open the test file in your favorite editor and refactor your existing code:**

```
var assert = require('assert');
var expect = require('chai').expect

describe('The Rocket', function() {
    it('should transport the crew to the moon', function() {
        expect(rocketname).to.be.a('string');
        expect(rocketname).to.equal('Saturn V');
    });
});
```


Do we *really*
need all of that?
© NASA

Yes.

1. **You will need Mocha to write your unit tests**
2. **You can use Chai to write your assertions in BDD fashion**
3. **You need a separate directory**
   In order to keep your project structure clean, conventional and understandable for co-workers
4. **You need a separate file for each test subject**
   If your app consists of several “parts” make sure each part has their own set of unit tests (e.g. Backend vs. Frontend)


OK? Go!

DEMO TIME

https://github.com/supernoir/firststeps


Yay! It works!


Staying in touch

Integration into existing Processes

1. **You can use npm, grunt, gulp and/or webpack to run tests**
2. **Automate a build process with integrated tests**
   Only deploy a branch if the tests pass
3. **Think ahead, BDD-style**
   Write your tests before you code, describing your intent before finding a suitable solution.
4. **Automate testing**
   and let the computer do your work!

Further Reading

1. **Check out the Documentation & Guides**
   https://mochajs.org
   http://chaijs.com/

2. **Watch some Tutorials**
   Pluralsight (pluralsight.com/courses/unit-testing-nodejs)
   YouTube (Jesse Warden's Channel)

3. **Read a book**
   Test-driving JavaScript Applications (Amazon, eBook)


Go beyond

Other JS Testing Frameworks

1. **Karma** (Test Runner)
2. **Jasmine** (BDD Framework)
3. **Jest** (Framework using Snapshots)
4. **Enzyme** (Testing utilities for React)
5. **AVA** (Concurrent tests)
6. *And (as always with JS) many, many more!*


Thank you!