Best Practices:
Testing Node.js for Stability
and Project Success

Walter Scarborough
SEA 2014
Congratulations!
You've decided to build a node.js webservice/webapp!
OR
Congratulations!

Your latest and greatest node.js webservice/webapp works!

Image from http://www.pxleyes.com/photoshop-picture/4a3be022a6a4b/Remote.html
...but how long will it keep working as you update it?
Don’t leave things to chance!
Set your project up for success at any stage.

- Set up a good dev environment
- Write testable code
- Write tests
• Technical Hurdles
  ▪ Javascript presents some unique language issues.
  ▪ How do we test network code?

• Project Hurdles
  ▪ How do we test backend services that are still under development?
  ▪ How do we organize all of this anyway?
Tools of the trade

- Testing
  - Linting - JSHint
  - Promise Library - Q.js
  - Testing Framework - Mocha.js
  - Assertion Library - Should.js or Chai.js
  - Network Mocks - Nock.js
"JavaScript is a language with more than its share of bad parts."

- Douglas Crockford, JavaScript: The Good Parts

Common Mistakes:

- not using var
- type coercion
- if without parens (blockless statements)
- eval
- typed wrappers

Full list: http://www.jslint.com/lint.html
Solution: Javascript linting programs

"Warning: JSLint will hurt your feelings."

- JSLint homepage http://www.jslint.com/lint.html

Image from
https://www.flickr.com/photos/pasukaru76/9824401426/in/photostream/
The first step in testing is to write testable code.

Testable code is

- Modular
- Readable

Some good references on the topic:

- Code Complete by Steve McConnell
- Test Driven Development by Kent Beck
Node.js uses callbacks.

Too many callbacks can make code difficult to test.
Nested callbacks often lead to this pyramid:

```javascript
Controller.createUser = function(request, response) {
  service.checkUsername(request.username, function(error, results) {
    if (error) {
      responseController.sendError(error.message, response);
    } else {
      service.createUser(request.username, function(error, newUser) {
        if (error) {
          responseController.sendError(error.message, response);
        } else {
          service.createProfile(newUser, function(error, profile) {
            if (error) {
              responseController.sendError(error.message, response);
            } else {
              // ...
            }
          });
        }
      });
    }
  });
};
```

It’s difficult to read, maintain and test.
Javascript promises can make life a lot simpler:

```
Controller.createUser = function(request, response) {
    service.checkUsername(request.username)
        .then(function() {
            return service.createUser(request.username);
        })
        .then(function(newUser) {
            return service.createProfile(newUser)
        })
        .then(function(profile) {
            // ...
        })
        .fail(function(error) {
            apiResponseController.sendError(error.message, response);
        });
};
```

The Q library is a great choice for node.js:
http://documentup.com/krisbowal/q/
Mocha is a lightweight javascript testing framework

- It can be run on individual files or an entire project
- It can be run from npm if a make file is set up and specified in package.json
- It can use a variety of assertion libraries

```javascript
describe("donut model", function() {
  it("should contain chocolate donuts", function() {
    var donut = new Donut();
    donut.flavor.should.equal('chocolate');
  });
});
```
Nock.js

"Nock is an HTTP mocking and expectations library for Node.js"
- https://github.com/pgte/nock

```javascript
varnock=require('nock');
nock('https://www.donut.com')
  .get('/flavors')
  .reply(200, JSON.stringify(['chocolate','vanilla']))
```

TACC
Let’s take this one step further with reusable mocks and data fixtures.

```javascript
DataFixture.flavors = ['chocolate', 'vanilla'];

DonutMocks.getFlavors = function(nock) {
  nock('https://www.donut.com')
    .get('/flavors')
    .reply(200, JSON.stringify(dataFixture.flavors))
  ;
  return nock;
};

var nock = require('nock');

describe("donut model", function() {
  it("should fetch flavors", function() {
    var donut = new Donut();
    donutMocks.getFlavors(nock);

    donut.getFlavors()
      .then(function(flavors) {
        flavors[0].should.equal('chocolate');
      });
  });
});
```
Project Test Organization

- myProject/tests - Project tests
- myProject/tests/fixtures - Common requests/responses for your mocks
- myProject/tests/mocks - Reusable networking mock objects
Solution Review

- Lint your code
- In general, use promises instead of callbacks
- Write testable code
- Use mock networking objects
- Organize common requests/responses into data fixtures
- Write tests
Questions?

wscarbor@tacc.utexas.edu