NODEJS, UNIT TESTING, TIPS, AND TRICKS

OR, LESSONS LEARNED AFTER DOING NODE FOR A WHILE COMING FROM A C# BACKGROUND

Brown bag

DEPENDENCY INJECTION

```
//getFileContents.js
var fs = require('fs');

module.exports = function(fileName, callback){
  fs.readFile(fileName, 'utf-8', callback);
}
```

IF I INJECT THE DEPENDENCY?

```
module.exports = function(fs, fileName, callback){
  fs.readFile(fileName, 'utf-8', callback);
}
```

IF I MAKE IT NICE?

```
module.exports = function(fs){
  return function(fileName, callback){
    fs.readFile(fileName, 'utf-8', callback);
  };
};
```

NOW, I CAN CREATE A FACTORY, YAY

```
//castle.cs.js
var fs = require('fs');
var GetFileContents = require('./getFileContents.js');

var Factory = function(){
  return {
    getFileContents: new GetFileContents(fs)
  };
};

var dependencies = new Factory();
```

NOW I CAN SHARE DEPENDENCIES IN A COOL WAY

```
module.exports = function(dependencies){
  dependencies.getFileContent('hello.js', function(){
    // ....
  });
}
```

NOW I "MADE MY CODE MORE TESTABLE"

```
var sinon = require('sinon');
var expect = require('chai').expect;
var GetFileContents = require('../getFileContents.js');

describe('when using getFileContents', function(){

  var fsMock = {
    readFile: sinon.stub().yields(null, 'file content');
  };
  var getFileContents = new GetFileContents(fsMock);

  it('should make a call to fs', function(){
    expect(fsMock.readFile.called).to.be.true;
  });
});
```

NOPE

NOPE

NOPE

NODE.JS IS SIMPLER THAN THAT: INJECTR

THIS WAS MY FILE

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

module.exports = function(fileName, callback){
  fs.readFile(fileName, 'utf-8', callback);
}
```

THIS IS MY TEST

```
var sinon = require('sinon');
var expect = require('chai').expect;
var injectr = require('injectr');
describe('when using getFileContents', function(){
  var fsMock = {
    readFile: sinon.stub().yields(null, 'file content');
  };
  var getFileContents = injectr('../getFileContents.js', {
    fs: fsMock
  });
  it('should make a call to fs', function(done){
    getFileContents('hello.txt', function(){
      expect(fsMock.readFile.called).to.be.true;
      done();
    });
  });
```

INJECTR

It is based on the nodejs' vm native module

- vm allows to "eval" your code in a clear context
- It wraps the "require" function
- You can override just some of the required dependencies
- Forces you to avoid using globals
- You can still inject the globals if you really need to

USAGE EXAMPLE

```
var fileToTest = injectr('path/to/file.js', {
//dependencies
underscore: underscoreStub,
'./some-internal-dependency': {}
},{
// if you need them, globals
  console: console,
  '__dirname': 'blablabla'
});

// then, your test...
```

WHAT IF

I STILL NEED TO SHARE A SINGLETON INSTANCE BETWEEN MY MODULES

FACTORY?

NOPE

INSTEAD:

You can wrap your dependency into a singleton constructor, and then just require it

CODE EXAMPLE

Look at that!

CONCLUSIONS

Nodejs is simple

- don't make it complex. better is: less files, less code
- If you are writing code that requires an IDE to be navigated, you are probably doing it wrong
- It is not about different patters, they just apply differently
- Sinon.js is a super cool library to do spies, mocks, stubs, etc (client-side too)
- Mocha has a watcher. When you do TDD you can keep mocha in a separate screen and that's enough

USEFUL STUFF:

- npm install injectr
- npm install sinon
- npm install chai
- npm install -g mocha