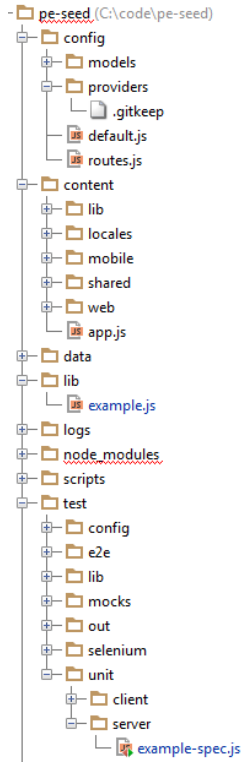
Server-side Unit Tests

Overview

- Examination of basic server-side testing strategies
  - Stubs and Spies
  - Dealing with promises
  - Mocking required modules
  - Async Tests
- Not example of what to test or test best practices
- Not example of logging or error handling best practices
- Not an explanation of sinon, chai, mocha, etc.
  - RTM

Directory Structure

- Libraries in
  - config/providers
  - lib
- Tests in
  - test/unit/server



Test Packages

- Mocha
- Sinon
- Chai
- Sinon-chai
- Sandboxed-module

Library to Test

- Post() sends data via POST to server
  - server in config
- Data is array of random characters
  - generated by node-random
- Utilizes Proj-evo core logger and rest client

```
var     pec  = require('projevo-core')
    ,   rest = pec.RestClient
    ,   log  = pec.Logger.logger()
    ,   config = require('config')
    ,   random = require("node-random")
    ,   Q = require("q");

function randomPoster () {
    this.Post = function() {
        var defer = Q.defer();
        random.strings({
            "length": 1,
            "number": 20,
            "upper": false,
            "digits": false
        }, function(error, data) {
            if (error) throw error;
            rest.request('POST',config.randomServer,data,{json : true}).then(function(res){
                log.info("POST was successful");
                defer.resolve();
            }).fail(function(err){
                log.error(err);
                defer.reject(err);
            })
        });
        return defer.promise;
    }
};
module.exports = new randomPoster();
```

What will be tested

POST is to correct address

POST data is what was generated

Successful posts are logged at info level

Failed posts are logged at error level

What will be needed for Tests

- Mocks
  - node-random
  - config
  - Evo
    - Logger
    - Rest Client
      - Promise-based
- Must deal with Async nature of library
  - library returns a promise

Best Advice for Testing: Start Simple

- Sanity test to get setup
  - require test packages
    - make sure in package.json
  - configure test packages
    - should syntax optional
  - test the existence of library
    - note use of “require”
      - not sandboxed!

```
"use strict"

var chai     = require("chai"),
    expect   = chai.expect,
    sinon    = require('sinon'),
    sinonChai = require('sinon-chai'),
    should   = chai.should(),
    sandbox = require('sandboxed-module');

    chai.should();
    chai.use(sinonChai);

    process.setMaxListeners(0); // avoid warnings

    var env;

    describe("Initial Sanity Test",function(){
        beforeEach(function() {
            env = {};
            env.poster = require("../../../lib/example.js");
        });
        it("should exist",function() {
            env.poster.should.exist;
        })
    });
```

Creating Basic Mocks/Stubs/Spies

- Understand how the library consumes the mock
- Mimic the necessary properties/methods
- Use stub()/spy() to test the interaction

Example Core.Logger

Logger in the application

```
,    log  = pec.Logger.logger()
log.info("POST was successful");
      log.error(err);
```

Mock logger in test

```
mockLogLibrary = {};
infoSpy = sinon.spy();
errorSpy = sinon.spy();
mockLogLibrary.logger = function() {
    return {
        "error" : errorSpy,
        "info" : infoSpy
    }
};
pec = {};
pec.Logger = mockLogLibrary;
```

- Core must return object with logger() method
- Calling logger must return an object with info/error methods
- info/error should be spies

Mocking a Promise

To a consuming library, a promise looks like a function that returns an object and an optional value to calling function. The returned object also has the ability to return an object and an optional value. These returned objects usually need to support then() and fail() methods that take callbacks. To mock a success you must fire the then() method so that its callback fires. Fire the fail() method to mock a rejection.

Testing a Promise with Mocha

Using a hook like “before” or “beforeEach” you must pass in “done” to the hook. You then call done() in either the then() or the fail(). Test as normal without passing in “done” to any tests.

Note: This assumes you mocked all async activities in the library so they return immediately. If not then you can use chai-as-promised. That is beyond the scope of this presentation.

RestClient.request Success Mock

```
var pec, mockPromise,mockRequest;
pec = {};
mockPromise = {};
mockPromise.then = sinon.stub().yields().returns({"fail" : sinon.stub()});
mockRequest = {};
mockRequest.request = sinon.stub().returns(mockPromise);
pec.RestClient = mockRequest;
```

- request returns promise
- promise has then() stub
  - yields() invokes callbacks passed to then()
  - Can pass value to yields() if expected
- then() returns object with fail() stub

RestClient.request Failure Mock

```
pec = {};
mockPromise = {};
mockFail = sinon.stub().yields("This is an error!!!");
mockPromise.then = sinon.stub().returns({"fail" : mockFail});
mockRequest = {};
mockRequest.request = sinon.stub().returns(mockPromise);
pec.RestClient = mockRequest;
```

- request returns promise
- promise has then() stub
- then() returns object with fail() stub
  - yields() calls fail() simulating rejection

Injecting Mocks

Required modules in library

```
var     pec    = require('projevo-core')
    ,   rest   = pec.RestClient
    ,   log    = pec.Logger.logger()
    ,   config = require('config')
    ,   random = require("node-random")
    ,   Q = require("q");
```

Sandboxed modules in test

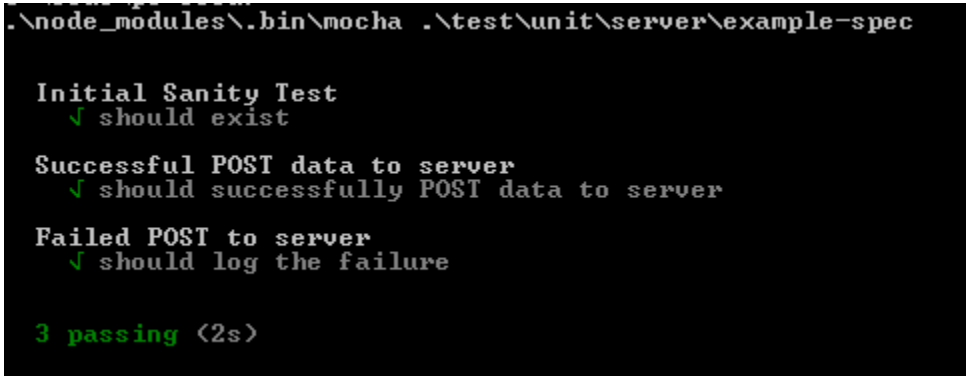
```
sandbox = require('sandboxed-module');

env.poster = sandbox.require("../../../lib/example.js", {
    requires : {
        'projevo-core' : pec,
        'config' : config,
        'node-random' : mockRandom
    }
});
```

The path to the library is passed as the first parameter to the sandbox.require call.
The string passed to “require” in library is passed as the key to an object in the second parameter of the sandbox.require call. The value of each corresponding key is a mock to inject. Anything not injected will default to the module referenced in the library’s require statement.

Testing a Promise

```
describe("Async Test",function(){
   beforeEach(function(done){
     env = {};
     env.poster = require("../../../lib/example.js");        env.poster.Post().then
     (function(){
           done()
       }).fail(function (err) {
           done()
       });
   });
  it("should do something",function(){
     expect(true).to.equal(true);
  })
});
```

Putting it All Together



https://github.com/CommandAlkon/pe-seed/tree/server-testing-demo/test

Source Code