Testing Java Script

Objectives

- Why Unit Test?
- How do I Unit Test in JavaScript
- Several frameworks and tools
  - QUnit
  - Jasmine
  - Sinon
  - Karma

Test Driven Development

- For each new feature write a test first
  - Define code requirements before writing code
  - Test based on use case
  - Acceptance test to test the big picture
  - Unit test to test the implementation
- Paradigm shift
  - From “what code must I write to solve this problem?”
  - To “how will I know when I’ve solved this problem”

Red, Green, Refactor

- Write test to express how code is used and what it must do
  - This test will fail
- Write enough code to pass the test, no more
- Re-factor
  - Clean code to remove redundancy & improve design
  - Re-run tests to make sure you didn't break anything
- Repeat until done
  - Write another test, push functionality forward

Benefits

- Code without tests risks being defective
- TDD guarantees high degree of test coverage
  - Test written as each feature added
- Reduces uncertainty
  - You can *prove* your code works successfully
- Reduces cost of bugs
  - Prevents bugs – less debugging later
  - Bugs prevented or found early save time later

What is a UnitTest?

- **Michael Feathers wrote "A set of Unit Testing Rules"**

A test is not a unit test if:
• It talks to the database
• It communicates across the network
• It touches the file system
• It can't run at the same time as other tests
• You have to do special things in your environment (such as editing config files) to run it.

Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes

Tools

- Many tools available
  - qUnit
  - Buster
  - Testacular/Karma
  - TestSwarm
  - JSTestDriver
  - YUI Yeti
  - Jasmine
  - Sinon

QUnit

- Originally written as the jQuery test library
  - Now spun off as its own project
  - no dependencies on jQuery

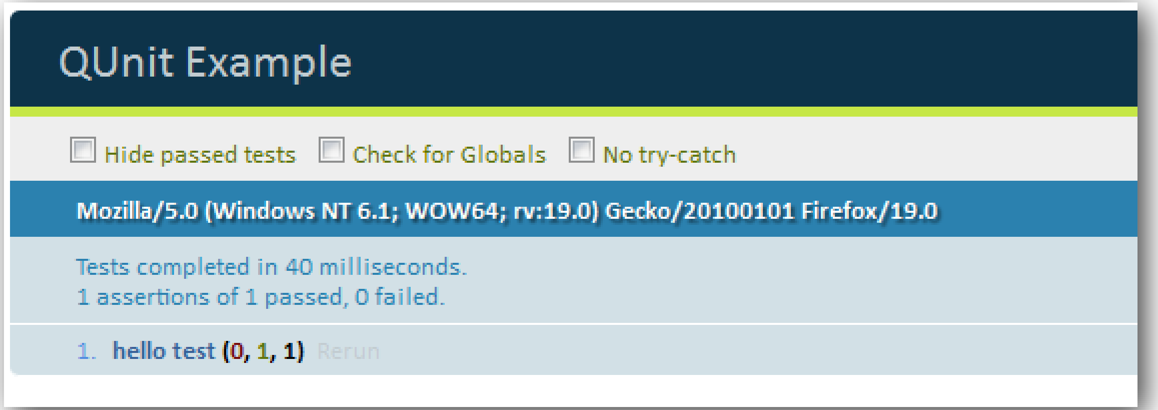
Initial Setup

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>QUnit Example</title>
    <link rel="stylesheet" href="resources/qunit.css">
</head>
<body>
<div id="qunit"></div>
<div id="qunit-fixture"></div>
<script src="resources/qunit.js"></script>
<script src="tests/tests.js"></script>
</body>
</html>
```

First Test

```
test( "hello test", function() {
    ok( 1 == "1", "Passed!" );
});
```

First result



QUnit Example

☐ Hide passed tests ☐ Check for Globals ☐ No try-catch

```
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0

Tests completed in 40 milliseconds.
1 assertions of 1 passed, 0 failed.

1. hello test (0, 1, 1) Rerun
```

QUnit Methods

- ok(truthy [, message])
  - Evaluate the first argument
- equal(actual, expected [, message])
  - Use == to compare actual and expected
- deepEqual(actual, expected [, message])
  - Use === to compare actual and expected

Grouping tests

- Want to logically organize tests
  - Can add a module to the tests

```
module( "group a" );

test( "first test", function() {
    ok( 1 == "1", "Passed!" );
});

test( "second test", function() {
    ok( 2 == "2", "Passed!" );
});
```

Common Code

- Want to logically organize tests
  - Can add a module to the tests

```
module( "group a" ,{
    setup: function(){
    }
});

test( "first test", function() {
    ok( 1 == "1", "Passed!" );
});

test( "second test", function() {
    ok( 2 == "2", "Passed!" );
});
```

Using test doubles

- Replacements for part of your code
  - Depended on Components (DOC) of SUT
  - Double provides same API as DOC

Why doubles?

- Return values of components may not be repeatable
  - Date/time values
- Calls may be 'risky' or may be charged for
  - Calling live web services during test
- Parts of the application are 'slow'
  - Database access
  - File access
- Unit tests should not rely on external resources
  - Databases
  - Web Services

Mocks, fakes, spies, stubs

- Fake Object
  - Provides same implementation as DOC but is much simpler
- Test Stub
  - Used to specify control options for SUT
  - e.g. different return values force SUT down different paths
- Test Spy
  - Like stub but also captures outputs of SUT
- Mock Object
  - Provides behaviour verification
  - e.g. correct methods called in correct order

JavaScript doubles

- Can simply replace methods on an instance
  - There's no need for a library
- However a library can help
  - Eases the repetitive tasks

Sinon

- **Sinon is a JavaScript mocking library**

> Standalone test spies, stubs and mocks for JavaScript.
> No dependencies, works with any unit testing framework.

With Sinon you can

- Write spies
- Write stubs
- Test Ajax
- Test XMLHttpRequest
- Test timers

Sinon Spies

- Spies used to show that a callback executes

```
var blogPosts = function() {
    getPosts = function (callback) {
        if(callback != null){
            callback();
        }
    }
    return {
        getPosts: getPosts
    }
}();
```

```
test("callback is called", function () {
    var callback = sinon.spy();
    blogPosts.getPosts(callback);
    ok(callback.called, "Callback called")
});
```

Testing Ajax

```
getPosts = function (callback) {
    $.ajax({
        url: "/blog/posts",
        success: function (data) {
            callback(data);
        }
    });
}
```

```
test("callback is called", function () {
var callback = sinon.spy();
sinon.stub(jQuery, "ajax").yieldsTo("success", [1, 2, 3]);
blogPosts.getPosts(callback);
ok(callback.called, "Call the callback ")});
ok(jQuery.ajax.calledWithMatch({url: "/blog/posts"}))
ok(callback.calledWith([1,2,3]))
//sinon.restore(jQuery.ajax);
```

Replacing XHR

```
var xhr, requests;
module("Test with XHR", {
    setup: function () {
        xhr = sinon.useFakeXMLHttpRequest();
        requests = [];
        xhr.onCreate = function(req) { requests.push(req); };
    },
    teardown: function(){
        xhr.restore();
    }
})

test("xhr is called", function () {
    var callback = sinon.spy();
    blogPosts.getPosts(callback);
    equal(requests.length, 1);
    equal(requests[0].url, "/blog/posts");
});
```

Replacing XHR – Sending Response

```
var xhr, requests;
module("Test with XHR", {
    setup: function () {
        xhr = sinon.useFakeXMLHttpRequest();
        requests = [];
        xhr.onCreate = function(req) { requests.push(req); };
    },teardown: function(){
        xhr.restore();
    }
})

test("callback is called", function () {
    var callback = sinon.spy();
    blogPosts.getPosts(callback);
    requests[0].respond(200,
                        {"Content-Type": "application/json"},
                        JSON.stringify({foo: "bar"}));
    ok(callback.called, "Call the callback")
});
```

Jasmine

- A BDD tool for JavaScript
  - Behaviour Driven Development
  - 'describe' the 'expectations' of the code
  - defined in English

Setup Jasmine

- Include the correct libraries

```
<script type="text/javascript" src="spec/SpecHelper.js"></script>
   <script type="text/javascript" src="spec/SlipsServiceSpec.js"></script>
   <script type="text/javascript" src="spec/ControllersSpec.js"></script>

   <script type="text/javascript">
       (function () {
           var jasmineEnv = jasmine.getEnv();
           ...
```

Test suite

- Describe the test

```
describe("RSK.Services.Entries", function () {


});
```

Specifications

- Specifies what the code should do
    - Use the Jasmine 'it' function

```
describe("RSK.Services.Entries", function () {
   it("is defined", function () {
        expect(RSK.Services.Entries).toBeDefined();
    });
});
```

Expectations

- If these are met the test has passed
  - Jasmine 'expect' function

```
describe("RSK.Services.Entries", function () {
   it("is defined", function () {
        expect(RSK.Services.Entries).toBeDefined();
    });

    it("returns data", function () {
        var slipsService = new RSK.Services.Entries.slipsService(http);
        var data = slipsService.get(new Date());
        expect(data.length).toBe(2);
    });
});
```

Setup and teardown

- **beforeEach and afterEach**

```
describe("RSK.Services.Entries", function () {
    beforeEach(function() {
        localize = {};
        scope = {};
        location = {};
        authenticate = {};
    });

    it("is defined", function () {
        expect(RSK.Services.Entries).toBeDefined();
    });

    it("returns data", function () {
        var slipsService = new RSK.Services.Entries.slipsService(http);
        var data = slipsService.get(new Date());
        expect(data.length).toBe(2);
    });
});
```

Nested describes

- **Allows better descriptions of the test output**

```
describe("RSK.Services.Entries", function () {
    it("is defined", function () {
        expect(RSK.Services.Entries).toBeDefined();
    });

    describe("slipsService", function () {
        it("is defined", function () {
            expect(RSK.Services.Entries.slipsService).toBeDefined();
        });
    });
});
```

Jasmine spies

- Jasmine offers its own 'spy' syntax
  - spyOn
  - createSpy
  - andReturn
  - andCallThrough

Karma

- Test runner
  - Install in Node.js
  - Monitors file system and runs test continually
  - Works with any test framework
    - requires an adapter
  - Works with multiple browsers

Installing Karma

- Karma is a Node application
  - Install node first (http://nodejs.org/download/)
  - the use npm to install Karma as a global module

```
$ npm install -g karma
```

Karma configuration

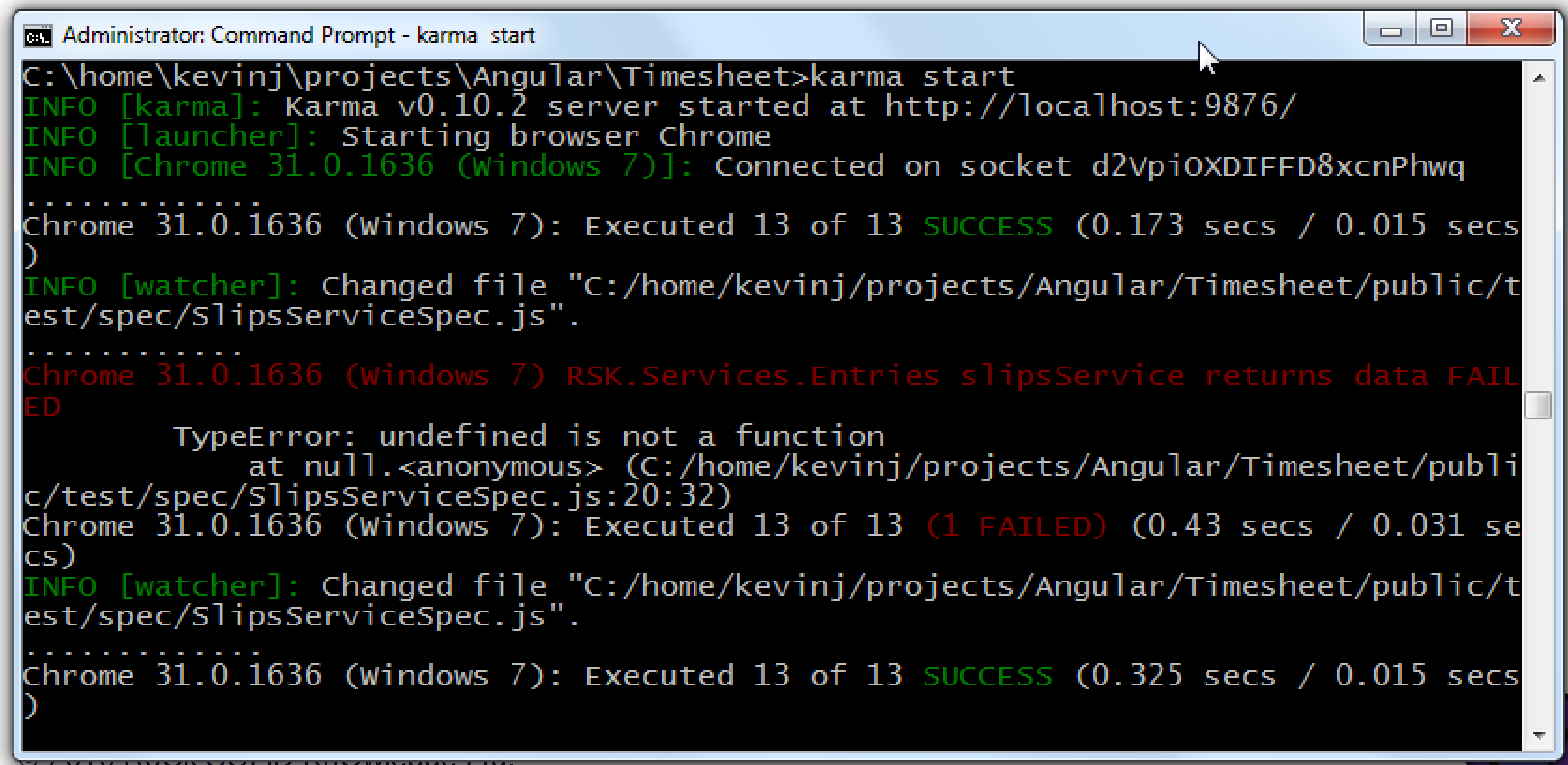
- Create a configuration file to run Karma
  - Which browsers
  - Which test framework
  - Which tests

```
$ karma init myconfig.js
```

Running Karma

- karma start
  - Looks for karma.conf.js by default

```
$karma start mykarmaconf.js
```



Administrator: Command Prompt - karma start

```
C:\home\kevinj\projects\Angular\Timesheet>karma start
INFO [karma]: Karma v0.10.2 server started at http://localhost:9876/
INFO [launcher]: Starting browser Chrome
INFO [Chrome 31.0.1636 (Windows 7)]: Connected on socket d2VpiOXDIFFD8xcnPhwq
.............
Chrome 31.0.1636 (Windows 7): Executed 13 of 13 SUCCESS (0.173 secs / 0.015 secs
)
INFO [watcher]: Changed file "C:/home/kevinj/projects/Angular/Timesheet/public/t
est/spec/SlipsServiceSpec.js".
............
Chrome 31.0.1636 (Windows 7) RSK.Services.Entries slipsService returns data FAIL
ED
        TypeError: undefined is not a function
            at null.<anonymous> (C:/home/kevinj/projects/Angular/Timesheet/publi
c/test/spec/SlipsServiceSpec.js:20:32)
Chrome 31.0.1636 (Windows 7): Executed 13 of 13 (1 FAILED) (0.43 secs / 0.031 se
cs)
INFO [watcher]: Changed file "C:/home/kevinj/projects/Angular/Timesheet/public/t
est/spec/SlipsServiceSpec.js".
.............
Chrome 31.0.1636 (Windows 7): Executed 13 of 13 SUCCESS (0.325 secs / 0.015 secs
)
```

Summary

- Unit testing is necessary to help write defect free code
- QUnit is an easy to use testing tool
- Can use Sinon to fake calls
- Can use Sinon to spy on calls
- Jasmine lets you use a BDD style syntax
- Karma to run tests continually