Telerik Academy

Unit Testing in JavaScript

With Mocha and Karma

Telerik Software Academy

Telerik Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

- Unit Testing
  - Overview
  - Test-driven & behavior-driven development
- Mocha & Chai
  - Overview and installation
  - Running simple tests
- Mocha HTML Reporter and Karma
- Creating a test suites and specs

Telerik Academy

What is Unit Testing?

Telerik Academy

Unit Test – Definition

A unit test is a piece of code written by a developer that exercises a very small, specific area of functionality of the code being tested.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

Edsger Dijkstra, [1972]

Manual Testing

- You have already done unit testing
  - Manually, by hand
- Manual tests are less efficient
  - Not structured
  - Not repeatable
  - Not on all your code
  - Not easy to do as it should be

Unit Test – Example

```
function sum(numbers)
{
  var sum = 0;
  for (var i=0; i<numbers.length; i++)
    sum += array[i];
  return sum;
}

void testSum()
{
 if (sum([1,2]) != 3)
    throw new Error("1+2 != 3");
 if (sum([-2]) != -2)
    throw new Error("-2 != -2");
 if (sum([]) != 0)
    throw new Error("0 != 0");
}
```

Unit Testing – Some Facts

- Tests are specific pieces of code
- In most cases unit tests are written by developers, not by QA engineers
- Unit tests are released into the code repository (TFS / SVN / Git) along with the code they test
- Unit testing framework is needed
  - QUnit, Jasmine, Mocha

Unit Testing – More Facts

- All objects should be tested
- All methods should be tested
  - Trivial code may be omitted
    - E.g. property getters and setters
  - Private methods can be omitted
    - Some gurus recommend to never test private methods → this can be debatable
- Ideally all unit tests should pass before check-in into the source control repository

Why Unit Tests?

- Unit tests dramatically decrease the number of defects in the code
- Unit tests improve design
- Unit tests are good documentation
- Unit tests reduce the cost of change
- Unit tests allow refactoring
- Unit tests decrease the defect-injection rate due to refactoring / changes

Telerik Academy

Unit Testing in JavaScript

Telerik Academy

Unit Testing in JavaScript

- There are too many frameworks for unit testing for JavaScript
  - Some use TDD
  - Others use BDD
- Some of the frameworks:
  - TDD
    - QUnit, JsUnit & Mocha
  - BDD
    - Jasmine & Mocha

QUnit

- QUnit was developed to test jQuery
  - Developed by John Resig
- QUnit has the following features:
  - Uses test-driven development (TDD)
  - Has a lot of asserts to match every need
  - Can test async code

Jasmine

- Jasmine is an open-source testing framework
  - Can run in both the browser and on Node.js
  - Uses behavior-driven development
  - Widely used
  - Introduces in 2008
- Jasmine has the following features:
  - Easy-to-read (expressional) syntax
  - Testing async code
  - Spies (mocking objects and methods)
  - DOM testing

Mocha

- Mocha is the new kid on the block
  - Open source framework, introduces in 2012
  - Can run in both the browser and on Node.js
  - Plugins for test syntax, spies, etc…
- Mocha has the following features:
  - Easy-to-read (expressional) syntax
  - Testing async code
  - Supports both BDD and TDD
  - The most used plugin for syntax is Chai.js
  - The most used plugin for spies is Sinon

Telerik Academy

Testing with Mocha

Overview and Installation

Telerik Academy

Mocha Overview

- Mocha is a feature-rich framework for testing JavaScript
  - Run in both the browser and on Node.js
  - Can test async code
  - Compatible with Karma & other test runners
  - Pluggable
    - Different plugins to add even more features

Installing Mocha

- To start working with Mocha follow the steps:
  1. Get Mocha
     - Download mocha from GitHub
     - With bower
     - With NuGet
  2. Setup a reporter
     - HTML reporter
     - Karma reporter

```
$ bower intall mocha
```

```
PM> Install-Package MochaChaiBdd
```

Telerik Academy

Installing Mocha

- To start working with Mocha follow the steps:
  3. Select a plugin for the test syntax
     - Mostly used is chai.js

```
$ bower intall chai
```

  4. Start writing tests

```
describe('#sum', function () {
  it('when empty array, expect to return 0', function () {
    var actual = sum([]);
    expect(actual).to.equal(0);
  });
  it('when with single number, expect the number', function () {
    var number = 6;
    var actual = sum([number]);
    var expected = number;
    expect(actual).to.equal(expected);
  });
});
```

Telerik Academy

Running Tests with Mocha and Chai

Live Demo

Telerik Academy

Mocha Reporters

Where Mocha can report the result of the tests?

Telerik Academy

Mocha Reporters

- What is a reporter?
  - Reporter is the place where Mocha outputs the result from the unit tests
    - If the tests passed
    - Or if they failed
- Mocha has a lot of reporters:
  - http://visionmedia.github.io/mocha/#reporters
  - Good reporters are the HTML reporter, the Spec reporter and the Karma reporter

Telerik Academy

Mocha HTML Reporter

- The HTML reporter outputs in the browser
  - Needs an HTML template:

```
<!-- document start -->
<head>
  <link rel="stylesheet" href="mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <!-- include mocha.js and chai.js -->
  <script type="text/javascript">
    mocha.setup('bdd');
    expect = chai.expect;
  </script>
  <!-- import javascript files and test files -->
  <script type="text/javascript">
    mocha.run();
  </script>
<!-- document end -->
```

Telerik Academy

Mocha HTML Reporter

- The HTML reporter outputs in the browser
  - Needs an HTML template:

```
<!-- document start -->
<head>
  <link rel="stylesheet" href="mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <!-- include mocha.js and chai.js -->
  <script type="text/javascript">
    mocha.setup('bdd');
    expect = chai.expect;
  </script>
  <!-- import javascript files and test files -->
  <script type="text/javascript">
    mocha.run();
  </script>
<!-- document end -->
```

Include the Mocha styles

Telerik Academy

Mocha HTML Reporter

- The HTML reporter outputs in the browser
  - Needs an HTML template:

```
<!-- document start -->
<head>
  <link rel="stylesheet" href="mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <!-- include mocha.js and chai.js -->
  <script type="text/javascript">
    mocha.setup('bdd');
    expect = chai.expect;
  </script>
  <!-- import javascript files and test files -->
  <script type="text/javascript">
    mocha.run();
  </script>
<!-- document end -->
```

Mocha will report here

Telerik Academy

Mocha HTML Reporter

- The HTML reporter outputs in the browser
  - Needs an HTML template:

```
<!-- document start -->
<head>
  <link rel="stylesheet" href="mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <!-- include mocha.js and chai.js -->
  <script type="text/javascript">
    mocha.setup('bdd');
    expect = chai.expect;
  </script>
  <!-- import javascript files and test files -->
  <script type="text/javascript">
    mocha.run();
  </script>
<!-- document end -->
```

Setup Mocha

Telerik Academy

Mocha HTML Reporter

- The HTML reporter outputs in the browser
  - Needs an HTML template:

```
<!-- document start -->
<head>
  <link rel="stylesheet" href="mocha/mocha.css">
</head>
<body>
  <div id="mocha"></div>
  <!-- include mocha.js and chai.js -->
  <script type="text/javascript">
    mocha.setup('bdd');
    expect = chai.expect;
  </script>
  <!-- import javascript files and test files -->
  <script type="text/javascript">
    mocha.run();
  </script>
<!-- document end -
```

Run the tests with Mocha

Telerik Academy

Mocha HTML Reporter

Live Demo

Telerik Academy

Karma

Overview

Telerik Academy

Karma Overview

- Karma is a testing environment
  - Getting instant feedback
  - Can run in any browser environment
    - Even in phantom.js environment
- Karma can work with most of the testing frameworks
  - Mocha
  - Jasmine
  - QUnit

Installing Karma

- Karma is a Node.js package
  - Must have Node.js installed, to run Karma
- Install Node.js from http://nodejs.org/
  - There is an installer for every platform:
    - Windows
    - Linux
    - Macintosh
- Install Karma with

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

Telerik Academy

Setting up Karma Environment

- To setup karma environment for a project follow the steps:

1. Install karma, if you have not done so already
2. Install the necessary packages for the wanted test framework:

```
$ npm install karma-mocha karma-chai --save-dev
```

3. Install one or many browser environments

```
$ npm install karma-chrome-launcher --save-dev
```

```
$ npm install karma-phantomjs-launcher --save-dev
```

4. Run `$ karma init` and fill in the needed data
5. Start the Karma environment with `$ karma start`

Telerik Academy

Mocha in Karma Environment

Live Demo

Telerik Academy

Mocha with Chai

Adding assertion framework

Telerik Academy

Mocha with Chai

- Mocha is made pluggable
  - Can use almost any assertion framework
  - The most used is Chai.js
- Chai.js uses behavior-driven development
- To use Chai in Mocha, do the following:
  - Install Chai: `$ bower install chai`
  - Add chai.js to your reporter `<script src="…/chai.js">`
  - Set the global `expect` object to `chai.expect` `expect = chai.expect;`

Telerik Academy

Mocha Test Suites and Specs

Mocha Test Suites

- Mocha uses test suites to order the tests
  - Tests suites are created with the describe(name, callback) function
    - Provide a name of the test suite and a callback function

```
describe('#sum', function() {
  //here are the tests
});
```

- Test suites can be nested in one another

```
describe('Person', function() {
  describe('when initializing', …
  describe('when changing name', …
});
```

Mocha Specs

- Spects (tests) are contained in a test suites
  - Call the function it(name, callback)
    - Has a name and a callback:

```
describe('Person', function() {
  describe('when initializing', function(){
    it('with valid names, expect ok', function(){
      var person = new Person('Peter', 'Petrov');
      expect(person.firstname()).to.equal('Peter');
      expect(person.lastname()).to.equal('Petrov');
    });
  });
});
```

Telerik Academy

Mocha Test Suites and Tests

Live Demo

Telerik Academy

Chai Assertions

How to use Chai.js?

Chai Assertions

- Chai.js is a assertion framework
  - Allows to assert expected/actual in tests
- Chai.js has three assertion styles
  - Assert style

```
assert.equal(person.firstname(), 'Peter');
```

  - Expect style

```
expect(person.firstname()).to.equal('Peter');
```

  - Should style

```
person.firstname().should.equal('Peter');
```

Telerik Academy

Chai Expect Assertion Style

Using the BDD Styles

Telerik Academy

Chai Expect Assertion Style

- Chai.js expect assertion style has a fluent and expressive syntax:

```
expect(person.firstname()).to.equal('Peter');
expect(person).to.be.a('Person');
expect(person).to.not.be.undefined;
expect(person).not.to.be.null;
expect(person).to.be.ok;
expect(person).not.to.be.ok;
expect(function(){
  //without name
  new Person();
}).to.throw(ReferenceError);
```

Telerik Academy

Chai Assertions

Live Demo

Telerik Academy

Unit Testing in JavaScript

http://academy.telerik.com