Testing

In this lab, you will familiarize yourself with the module Mocha, a popular unit test framework for Node.

# Objectives

You will learn to use the Test Driven Development (TDD) interface of Mocha by creating three simple unit tests.

# Install Mocha

The first thing we have to do is to use Node Package Manager, npm, to install the 'mocha' module. Do this by opening a command prompt in the lesson directory and issuing the command npm install -g mocha. The –g flag installs the module in the global node\_modules and provides the mocha command line interface.

# Lab Steps

Now that Mocha is installed, we can begin the lab, whose goal is to create three unit tests for a simple Product class, which has been provided for you in product.js.

## Run the empty test suite

An empty test suite has been created for you in productTest.js. The Product class is instantiated and the Mocha suite definition is set up although it is currently empty.

1. Run the test suite by issuing the following on the command line: mocha productTest.js -u tdd. If everything is working correctly, you should see a message in the console indicating that there are 0 passing tests. This makes perfect sense because we do not yet have any tests created! Mocha defaults to Behavior Driven Development (BDD) syntax but for this lab, we will be using the TDD syntax, which we indicate by using the command line option “-u tdd”. The main differences in BDD and TDD are the syntax used to describe tests but functionally, the test cases should behave the same.

## Create the delete test case

2. Create the first unit test declaration with the signature test('deleteTest', function deleteTest() {…}). The first thing the unit test should do is add a product using the API. Use Node's assert module to make sure the product count is 1 by invoking assert.equal(). Then run the deleteAll API and again assert that the count is now 0.

3. Execute the unit test with the command line mocha productTest.js -u tdd and verify that there is one passing test.

## Create the add test case

4. Create another unit test declaration with the signature test('addTest', function addTest() {…}). Use the deleteAll API to reset the product list and then use the add API to create one. Use either assert.equal or assert.notEqual to test for the expected result and run your test suite again from the command line.

## Create the doAsync test case

Mocha allows you to test asynchronous code by adding a callback to the test function (usually named done). By adding the callback, Mocha knows that it should wait for completion. There is a 2 second timeout by default for a test case but this is configurable with a command line option: -t.

6. Create a unit test with the signature test('doAsyncTest', function doAsyncTest(done) {…}. The Product class contains a doAsync function which makes use of Node's setTimeout(callback, delay, args…) API. After waiting for 1.5 seconds, the value of true is returned. The test case should call the API with product.doAsync(function callback(value) {…} and assert that value is true. Then call the done() callback to indicate to Mocha that the test case is complete. If you modify the Product class to increase the timeout to a value greater than 2s, you can see that the test case fails because of Mocha's default timeout value.

Once you have created all three unit test cases and they are passing, you are complete with this lab!

The file productTest-bdd.js in the solutions folder is an example of tests done in Behavior-Driven Development (BDD) style.

A convenient mocha switch is “–w”: this will cause mocha to monitor the current working directory and rerun tests upon changes. Try it out!