Testing using Jasmine (Node.js) in JavaScript

Overview

In this tutorial, we will create some tests in JavaScript for Node.js, using jasmine-node.

"jasmine-node" is an implementation of Jasmine for Node.js and supports out-of-the-box JUnit reports.

Requirements

- nodejs
- "jasmine-node", "express" and "request" node modules

Description

This example is a simple Node.js web server tutorial from SemaphoreCI, based on the "express" node module.

It consists of two source files: one that basically returns an array of strings given a number, and another that implements the web server that processes an argument.

app/generator.js

```
 exports.generateHelloWorlds = function(number) {
  var result = [];
  for(var i=0; i < number; i++) {
    result.push("Hello World");
  }
  return result;
}
```

app/hello_world.js

```
var generator = require('./generator');
var express   = require('express');
var app = express();
app.get("/", function(req, res) {
  var number = req.query.number;
  var helloWorldArray = generator.generateHelloWorlds(number);
  res.send(200, helloWorldArray);
});
app.listen(3000);
```

package.json

```
{
  "name": "jasmine_helloworld_express",
  "version": "1.0.0",
  "description": "simple express hello world",
  "main": "index.js",
  "scripts": {
    "test": "node_modules/.bin/jasmine-node --junitreport --captureExceptions spec"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.1",
    "request": "^2.79.0"
  },
  "devDependencies": {
    "jasmine-node": "^1.14.5"
  }
}
```

There are two jasmine-node "specs" containing the tests that validate the previous implementations.

spec/generatorSpec.js

```
var generator = require("../app/generator");
describe("Hello World Generator", function() {
  it("returns an array", function() {
    expect(generator.generateHelloWorlds(0)).toEqual([]);
  });
  it("returns the correct number of Hello Worlds", function() {
    var result = generator.generateHelloWorlds(3);
    expect(result.length).toBe(3);
  });
  it("returns only Hello Worlds", function() {
    var result = generator.generateHelloWorlds(3);
    result.forEach(function(element) {
      expect(element).toBe("Hello World");
    });
  });
});
```

spec/helloWorldSpec.js

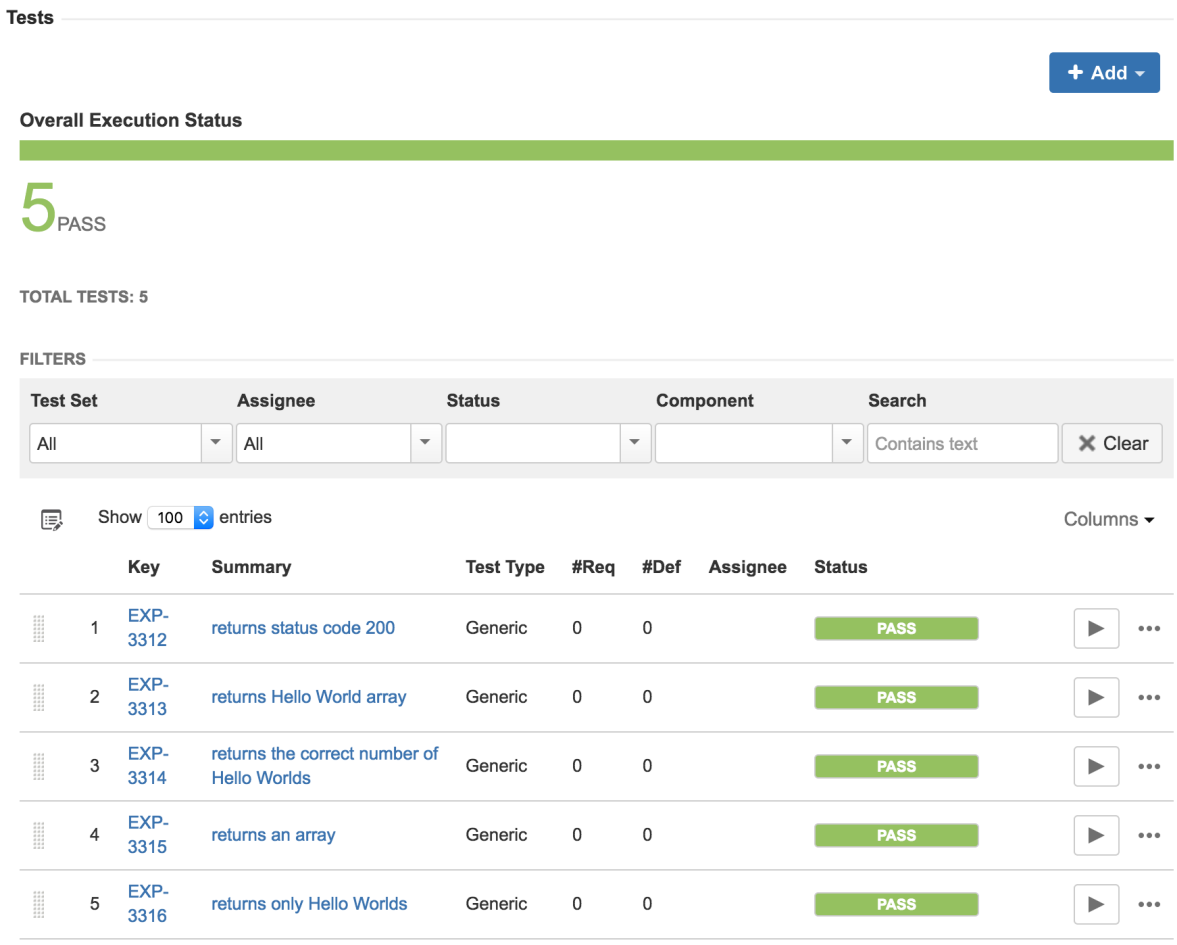
```
var request = require("request");
describe("Hello World Server", function() {
  describe("GET /", function() {
    it("returns status code 200", function(done) {
      request.get("http://localhost:3000", function(error, response, body) {
        expect(response.statusCode).toBe(200);
        done();
      });
    });
    it("returns Hello World array", function(done) {
      request.get("http://localhost:3000?number=3", function(error, response, body) {
        expect(body).toBe(JSON.stringify(["Hello World", "Hello World", "Hello World"]));
        done();
      });
    });
  });
});
```

In order to run the tests, we need first to run our basic web server in the background or in another console.

```
node app/hello_world.js
```

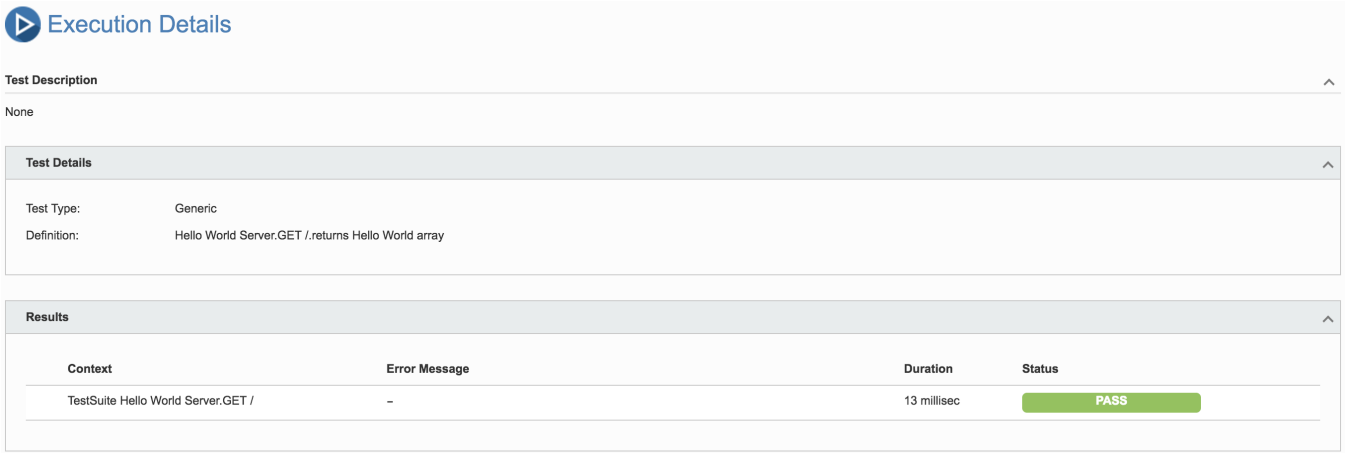
After running the tests and generating the JUnit XML reports (e.g., TEST-HelloWorldGenerator.xml, TEST-HelloWorldServer.xml), they can be imported to Xray (either by the REST API or through the **Import Execution Results** action within the Test Execution).

```
npm test
# jasmine-node --junitreport --captureExceptions spec
```



JUnit's Test Case is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the value of the "it" concatenated with the several "describe"' that make up the test case.

The Execution Details of the Generic Test contains information about the Test Suite, which in this case corresponds to the concatenation of the test's "describe".



References

- https://jasmine.github.io/
- https://semaphoreci.com/community/tutorials/getting-started-with-node-js-and-jasmine