JavaScript Test Driven Development

PhantomJS and Jasmine

Test Driven Development can be overwhelming. It sounds like a lot of hard, tedious, work when beginning to code. Some may say writing code to do nothing but test the code you’re going to use seems useless and a waste of time. However it is important to use TDD. It forces your brain to think! Writing test forces me to think about what exactly I am trying to achieve. I have to be specific about what I am trying to do. Once it’s clear then I can write the code. Practicing TDD encourages me to write tests and write smaller, easier-to-understand units of code.

Basically, your program will have a bunch of functions and classes. You want to make sure that, no matter what you throw at them, they'll perform how you want them to. For example, this function should always return a string that says "hello" in it. Testing ensures that everything goes down exactly how you planned.

What is Test Driven Development?

It is an approach to writing software where you write tests before you write application code. The basic steps are the following:

1. Write the test and make sure it fails.
2. Write the simplest, easiest possible code to make the test pass.
3. Refactor optimize or simplify the application code, making sure all the test still pass.
   1. To me this is the most important part. Hitting the test with everything you got, null , string, integer, and etc to ensure your test works.

Now for the fun part!!!

We are going to set up a simple Testing Application with PhantomJS and Jasmine

**PhantomJS**

PhantomJS is a headless Webkit scriptable with a JavaScript API. It is a light weight, fast and native support for various web standards: DOM handling, CSS Selector, JSON, Canvas, and SVG.

* Headless Website Testing: Run functional tests with frameworks such as Jasmine, Mocha and more. I use Jasmine.
* Page Automation: Access and manipulate webpages with the standard DOM API, or with the usual libraries like JQuery.
* Network Monitoring: Monitor page loading and export as standard HAR files(HTTP archive format for logging web browser’s interaction with a site).

**Jasmine**

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM.

**First lets talk about Jasmine**

Set up your Basic App

Create a directory that looks like this:

**-Parent**

**-JS**

**-phantomjs.js**

**-index.html**

You will need NodeJs to install the JavaScript libraries. Node is a Node Package Manager to install our JavaScript libraries. If you don’t have NodeJS go [here](https://nodejs.org/) to download and install it!

You will be using npm, so create a package.json file by typing:

**npm init**

It will walk you through creating a package.json file by asking several questions.

Install PhantomJS

**npm install phantomjs-prebuilt --save-dev**

Install Jasmine

**npm install jasmine --save-dev**

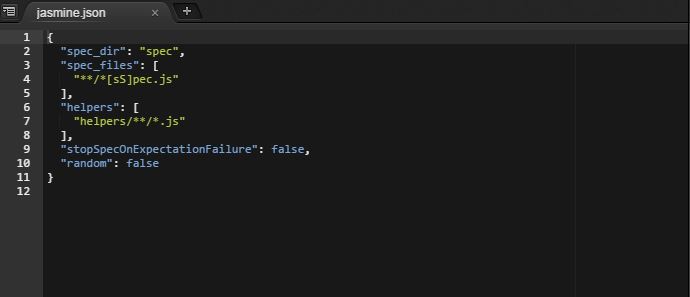
Install Jasmine Reporter (used to generate XML files of Jasmine testing logs)

**npm install jasmine-reporters --save-dev**

**initiate Jasmine**

: **jasmine**

This will create a spec folder. This folder will contain your testing files, jasmine.json file. The jasmine.json file tells jasmine that the tests are in a directory called spec and that each file contains tests begins with the word spec/Spec. It tells jasmine where the helper files are. There are options available to edit to.



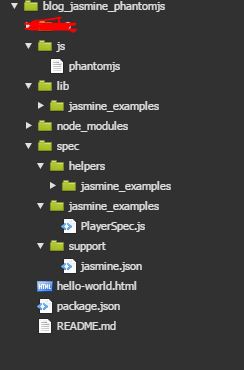
**Create jasmine examples**

**: jasmine examples**

This will create an example testing project. It will create a lib folder containg, jasmine\_examples folder. In the jasmine\_examples folder it will create a the example functions to test, Player.js and Song.js.

In the spec folder it will create the helpers folder and jasmine\_examples folder. Helpers /jasmine\_examples folder contains a SpecHelper.js file. This contains helper functions that assist in running your test functions. The spec/jasmine\_examples folder contains the testing file PlayerSpec.js. See how the file ends in Spec.js. The jasmine.json files tell jasmine to use this type of files to run tests.

You file directory should look like this:



**Initiate Jasmine Reporter**

Take the following code and place it at the top of the SpecHelper.js File.

var reporters = require('jasmine-reporters');

var junitReporter = new reporters.JUnitXmlReporter({

savePath: \_\_dirname,

consolidateAll: false

});

jasmine.getEnv().addReporter(junitReporter);

Type **Jasmine** in your terminal and see your tests run!

You should see the following:

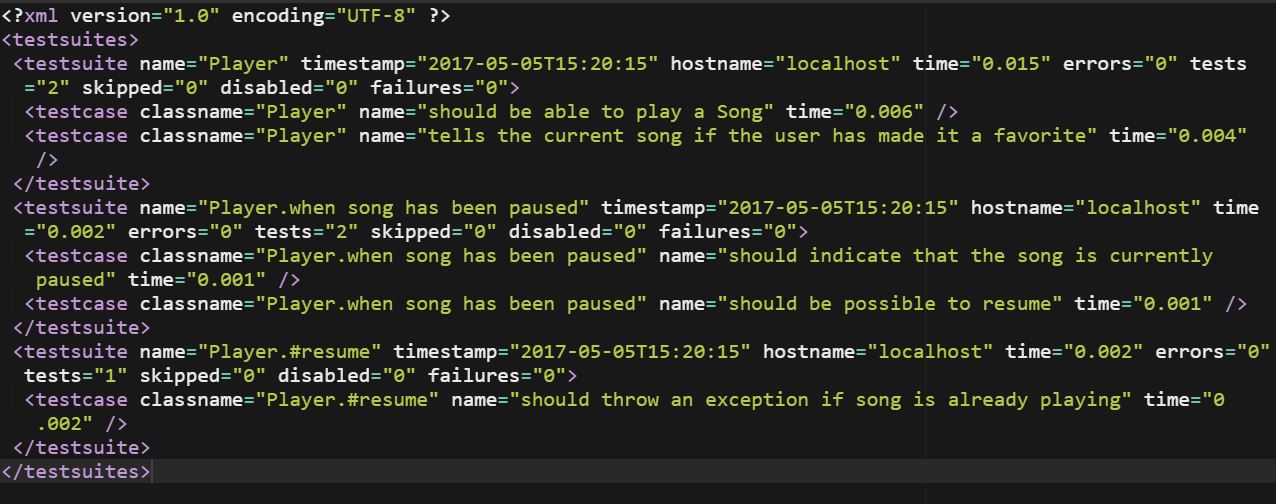
Started

.....

5 specs, 0 failures

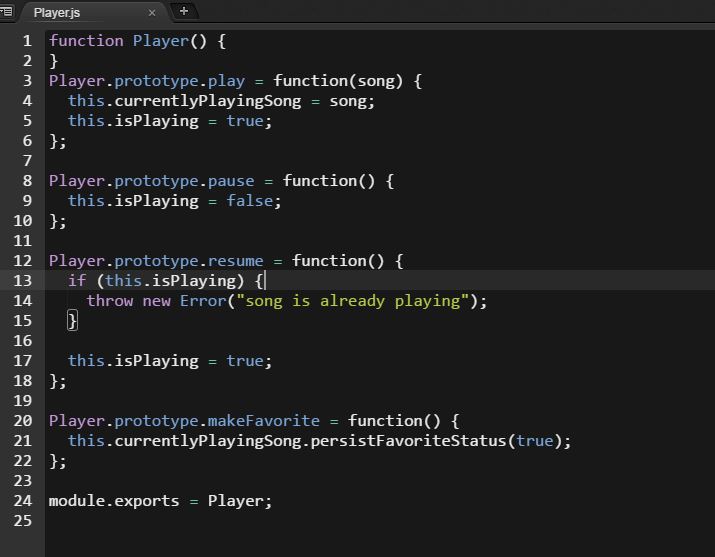
Finished in 0.018 seconds

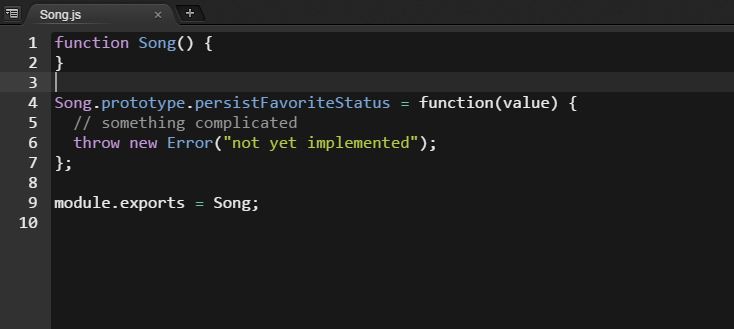
And you will see a juniresults-Player.xml file created with detailed testing results.



What is it doing:

First let’s start with our Test functions in Player.js and Song.js.





A function object is declared. Then the classes are created by using prototype property. There are two function objects Player() and Song(). Player() has 4 class functions and Song has 1 class function. Parent object function is encapsulated into a single unit of code by assigning that function object to module.exports. Now it can be publicly available to use in our testing js file, using require. We will talk about that later.

Now the tests:



All of our tests run using Node, which means we can use the node-style require() method. This just loads a javascript file or module into this PlayerSpec.js file. The next thing to know is that I amusing ‘Behavior Driven Development’ style to write the tests. This is a variation of TDD where are tests are written in the form: *Describe [****thing****] . It should [****do something****].* Jasmine includes built in functions like describe() and it() to make writing this style possible. The expect() chain does the checking.

describe(‘Player’) is called a **suite**. It is component of the application. This could be a class or just a slew of functions. The it() functions are called **spec.** This is the piece that tells what some piece of your program should do. Before each spec is called declare the player and song object. beforeEach() is a function that allows to run some shared setup before each of the specs in the suite, in which it is is called. The describe() is passed the string of the named function you are testing and the a function containing your specs. Inside the it() function the expect() chain does the checking.

Speaking of the it(‘should be able to play a song’) function example:

The expect() is passed the player.currentlyPlayingSong function. The chained toEqual() evaluates the variable and sees if it will pass the expect function test.

There is a lot more to learn. Go to jasmine.org to read up on it more.

Lets get into PhantomJS!

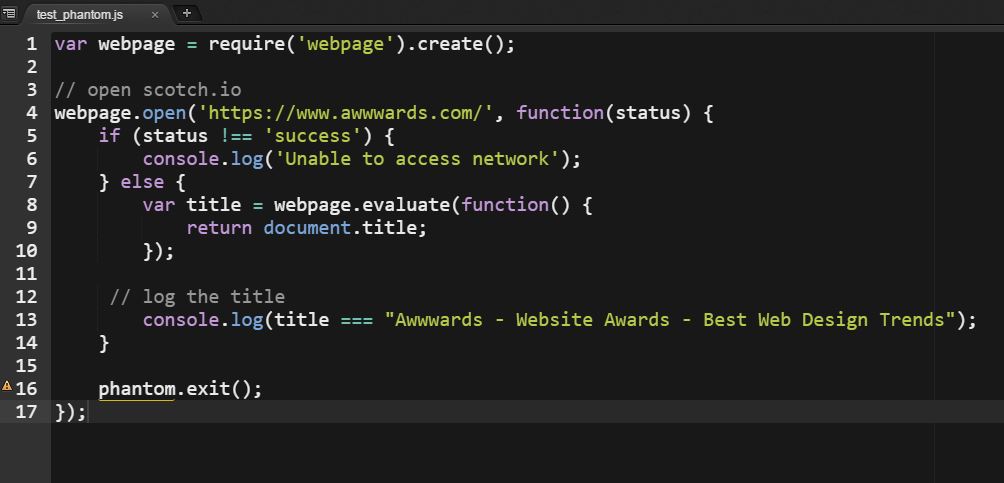
Like I said in the beginning PhantomJS is a headless WebKit scriptable with a Javascript API. In simple terms PhantomJS is a web browser without a graphical interface. PhantomJS is not for use in surfing the web. It has other cool features such as Page automation, Network Monitoring, Testing, Screen capture and more.

Page Automation

Page automation allows a developer to test the UI, making sure it works the way it was built to work.

This helps developers run a bunch of tests without ever having to open a web browser.

Although this may not seem important, this allows us to automate any sort of interactions with a web page without having to open a browser.



Copy and paste this code into the js/test\_phantom.js file

Run phantomjs <path to file>

This test checks to see if the document.title property of the website is correct.