Community Experience Distilled

Test your JavaScript applications efficiently using Jasmine and React.js

Jasmine JavaScript Testing
Second Edition

Paulo Ragonha

Take your testing of JavaScript applications to a new level of efficiency and reliability with the help of this book. Starting with the fundamentals of Jasmine and behavior-driven development (BDD), you will learn about testing and automation. You will learn how to create a sustainable codebase with the help of Jasmine. You will also take a look at integrated testing with React.js and Karma, and how you can speed this process up by faking AJAX requests. As you progress through the book, you will learn about the challenges of testing an application built on top of a framework and how you can prevent your application from suffering from dependency management. You will understand the concept of client-side and server-side rendering and test applications using React.js. You will also learn to automate using webpack and work with JavaScript module systems using ECMA Script 6 (ES6) modules.

By the end of this book, you will be a competent web developer with good knowledge and a strong understanding of how to efficiently use the Jasmine framework for testing purposes.

Who this book is written for
This book is for web developers and designers who work with React.js and JavaScript and who are new to unit testing and automation. It's assumed that you have a basic knowledge of JavaScript and HTML.

What you will learn from this book
- Understand and use the power of Jasmine to create better and more maintainable code bases
- Drive your application development entirely by tests
- Write modular and reusable code through the power of ECMA Script 6 (ES6) modules
- Use asynchronous tests, stubs, and spies optimally
- Test drive a React.js single-page application
- Optimize your code to unleash the power of tooling and automation

Jasmine JavaScript Testing
Second Edition

Test your JavaScript applications efficiently using Jasmine and React.js

Paulo Ragonha

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with Jasmine'
- A synopsis of the book’s content
- More information on Jasmine JavaScript Testing Second Edition

About the Author

Paulo Ragonha is a software engineer with over 7 years of professional experience. An advocate of the open Web, he is inspired and driven to build compelling experiences on top of this ubiquitous platform.

He loves to hack, so you will often see him wandering around in conferences or attending hackathons. His most recent professional experiences ranged from DevOps (with Chef and Docker) to moving up the stack with Node.js, Ruby, and Python and all the way toward building single-page applications (mostly with Backbone.js and "ad hoc" solutions).

Passionate about automation, he sees testing as a liberating tool to enjoy the craft of writing code even more. Back in 2013, he wrote the first edition of the book Jasmine JavaScript Testing, Packt Publishing.

Paulo has an amazing wife, who he loves very much. He lives in beautiful Florianópolis, a coastal city in the south of Brazil. He is a casual speaker, a biker, a runner, and a hobbyist photographer.
Jasmine JavaScript Testing
Second Edition

This book is about being a better JavaScript developer. So, throughout the chapters, you will not only learn about writing tests in the Jasmine 'idiom', but also about the best practices in writing software in the JavaScript language. It is about acknowledging JavaScript as a real platform for application development and leveraging all its potential. It is also about tooling and automation and how to make your life easier and more productive.

Most importantly, this book is about craftsmanship of not only working software, but also well-crafted software.

*Jasmine JavaScript Testing, Second Edition* is a practical guide to writing and automating JavaScript testing for web applications. It uses technologies such as Jasmine, Node.js, and webpack.

Over the course of the chapters, the concept of test-driven development is explained through the development of a simple stock market Investment Tracker application. It starts with the basics of testing through the development of the base domain classes (such as stock and investment), passes through the concepts of maintainable browser code, and concludes with a full refactoring to a React.js application build on ECMA Script 6 modules and automated build.
What This Book Covers

Chapter 1, Getting Started with Jasmine, covers the motivations behind testing a JavaScript application. It presents the concept of BDD and how it helps you to write better tests. It also demonstrates how easy it is to download Jasmine and start coding your first tests.

Chapter 2, Your First Spec, helps you learn the thought process behind thinking in terms of test-driven development. You will code your very first JavaScript functionality driven by tests. You will also learn the basic functions of Jasmine and how to structure your tests. Also demonstrated, is how Jasmine matchers work and how you can create one of your own to improve your tests' code readability.

Chapter 3, Testing Frontend Code, covers some patterns in writing maintainable browser code. You will learn about thinking in terms of components and how to use the module pattern to better organize your source files. You will also be presented with the concept of HTML fixtures and how you can use it to test your JavaScript code without requiring your servers to render an HTML. You will also learn about a Jasmine plugin called jasmine-jquery and how it can help you write better tests with jQuery.

Chapter 4, Asynchronous Testing – AJAX, talks about the challenges in testing AJAX requests and how you can use Jasmine to test any asynchronous code. You will learn about Node.js and how to create a very simple HTTP server to use as a fixture to your tests.

Chapter 5, Jasmine Spies, presents the concept of test doubles and how to use spies to do behavior checking.

Chapter 6, Light Speed Unit Testing, helps you to learn about the issues with AJAX testing and how you can make your tests run faster using stubs or fakes.

Chapter 7, Testing React Applications, introduces you to React, a library to build user interfaces, and covers how you can use it to improve the concepts presented in

Chapter 3, Testing Frontend Code, to create richer and more maintainable applications, of course, driven by tests.

Chapter 8, Build Automation, presents you with the power of automation. It introduces you to webpack, a bundling tool for frontend assets. You will start to think in terms of modules and their dependencies, and you will learn how to code your tests as modules. You will also learn about packing and minifying the code to production and how to automate this process. Finally, you are going to learn about running your tests from a command line and how this can be used in a continuous integration environment with Travis.ci.
Getting Started with Jasmine

It is an exciting time to be a JavaScript developer; technologies have matured, web browsers are more standardized, and there are new things to play with every day. JavaScript has become an established language, and the Web is the true open platform of today. We’ve seen the rise of single-page web applications, the proliferation of Model View Controller (MVC) frameworks, such as Backbone.js and AngularJS, the use of JavaScript on the server with Node.js, and even mobile applications created entirely with HTML, JavaScript, and CSS using technologies such as PhoneGap.

From its humble beginnings with handling HTML forms, to the massive applications of today, the JavaScript language has come very far, and with it, a number of tools have matured to ensure that you can have the same level of quality with it that you have with any other language.

This book is about the tools that keep you in control of your JavaScript development.

JavaScript – the bad parts

There are many complications when dealing with client JavaScript code; the obvious one, is that you cannot control the client’s runtime. While on the server, you can run a specific version of your Node.js server, you can't oblige your clients to run the latest version of Chrome or Firefox.

The JavaScript language is defined by the ECMAScript specification; therefore, each browser can have its own implementation of a runtime, which means there could be small differences or bugs between them.
Besides that, you have issues with the language itself. Brendan Eich developed JavaScript in just 10 days, under a lot of management pressure at Netscape. Although it got itself right in its simplicity, first-class functions, and object prototypes, it also introduced some problems with the attempt to make the language malleable and allow it to evolve.

Every JavaScript object is mutable; this means that there is nothing you can do to prevent a module from overwriting pieces of other modules. The following code illustrates how simple it is to overwrite the global `console.log` function:

```javascript
console.log('test');
>> 'test'

console.log = 'break';
console.log('test');
>> TypeError: Property 'log' of object #<Console> is not a function
```

This was a conscious decision on the language design; it allows developers to tinker and add missing functionality to the language. But given such power, it is relatively easy to make a mistake.

Version 5 of the ECMA specification introduced the `Object.seal` function, which prevents further changes on any object once called. But its current support is not widespread; Internet Explorer, for example, only implemented it on its version 9.

Another problem, is with how JavaScript deals with type. In other languages, an expression like `'1' + 1` would probably raise an error; in JavaScript, due to some non-intuitive type coercion rules, the aforementioned code results in `'11'`. But the main problem is in its inconsistency; on multiplication, a string is converted into a number, so `'3' * 4`, is actually `12`.

This can lead to some hard-to-find problems on big expressions. Suppose you have some data coming from a server, and although you are expecting numbers, one value came as a string:

```javascript
var a = 1, b = '2', c = 3, d = 4;
var result = a + b + c * d;
```

The resulting value of the preceding example is `'1212'`, a string.

These are just two common problems faced by developers. Throughout the book, you are going to apply best practices and write tests to guarantee that you don't fall into these, and other, pitfalls.
Jasmine and behavior-driven development

Jasmine is a little behavior-driven development (BDD) test framework created by the developers at Pivotal Labs, to allow you to write automated JavaScript unit tests.

But before we can go any further, first we need to get some fundamentals right, starting with what a test unit is.

A test unit is a piece of code that tests a functionality unit of the application code. But sometimes, it can be tricky to understand what a functionality unit can be, so for that reason, Dan North came up with a solution in the form of BDD, which is a rethink of test-driven development (TDD).

In traditional unit testing practice, the developer is left with loose guidelines on how to start the process of testing, what to test, how big a test should be, or even how to call a test.

To fix these problems, Dan took the concept of user stories from the standard agile construct, as a model on how to write tests.

For example, a music player application could have an acceptance criterion such as:

**Given a player, when the song has been paused, then it should indicate that the song is currently paused.**

As shown in the following list, this acceptance criterion is written following an underlying pattern:

- **Given**: This provides an initial context
- **When**: This defines the event that occurs
- **Then**: This ensures an outcome

In Jasmine, this translates into a very expressive language that allows tests to be written in a way that reflects actual business values. The preceding acceptance criterion written as a Jasmine test unit would be as follows:

```javascript
describe("Player", function() {
    describe("when song has been paused", function() {
        it("should indicate that the song is paused", function() {

        });
    });
});
```
You can see how the criterion translates well into the Jasmine syntax. In the next chapter, we will get into the details of how these functions work.

With Jasmine, as with other BDD frameworks, each acceptance criterion directly translates to a test unit. For that reason, each test unit is usually called a spec, short for specification. During the course of this book, we will be using this terminology.

**Downloading Jasmine**

Getting started with Jasmine is actually pretty simple.

Open the Jasmine website at [http://jasmine.github.io/2.1/introduction.html#section-Downloads](http://jasmine.github.io/2.1/introduction.html#section-Downloads) and download the **Standalone Release** (version 2.1.3 is going to be used in the book).

While at the Jasmine website, you might notice that it is actually a live page executing the specs contained in it. This is made possible by the simplicity of the Jasmine framework, allowing it to be executed in the most diverse environments.

After you've downloaded the distribution and uncompressed it, you can open the SpecRunner.html file on your browser. It will show the results of a sample test suite (including the acceptance criterion we showed you earlier):

```
5 specs, 0 failures  raise exceptions

Player
  should be able to play a Song
  when song has been paused
    should indicate that the song is currently paused
    should be possible to resume
  tells the current song if the user has made it a favorite
#resume
  should throw an exception if song is already playing
```

This shows the SpecRunner.html file opened on the browser.

This SpecRunner.html file is a Jasmine browser spec runner. It is a simple HTML file that references the Jasmine code, the source files, and the test files. For convention purposes, we are going to refer to this file simply as **runner**.
You can see how simple it is by opening it on a text editor. It is a small HTML file that references the Jasmine source:

```
<script src="lib/jasmine-2.1.3/jasmine.js"></script>
<script src="lib/jasmine-2.1.3/jasmine-html.js"></script>
<script src="lib/jasmine-2.1.3/boot.js"></script>
```

The runner references the source files:

```
<script type="text/javascript" src="src/Player.js"></script>
<script type="text/javascript" src="src/Song.js"></script>
```

The runner references a special SpecHelper.js file that contains code shared between specs:

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

The runner also references the spec files:

```
<script type="text/javascript" src="spec/PlayerSpec.js"></script>
```

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

The Jasmine framework is set up inside the lib/jasmine-2.1.3/boot.js file, and although it's an extensive file, most of its content is in documentation on how the setup actually happens. It is recommended that you open it in a text editor and study its content.

Although, for now, we are running the specs in the browser, in Chapter 8, Build Automation, we are going to make the same specs and code run on a headless browser, such as PhantomJS, and have the results written on the console.

A headless browser is a browser environment without its graphical user interface. It can either be an actual browser environment, such as PhantomJS, which uses the WebKit rendering engine, or a simulated browser environment, such as Envjs.

And although not covered in this book, Jasmine can also be used to test server-side JavaScript code written for environments such as Node.js.

This Jasmine flexibility is amazing, because you can use the same tool to test all sorts of JavaScript code.
Summary
In this chapter, you saw some of the motivations behind testing a JavaScript application. I showed you some common pitfalls of the JavaScript language and how BDD and Jasmine both help you to write better tests.

You have also seen how easy it is to download and get started with Jasmine.

In the next chapter, you are going to learn how to think in BDD and code your very first spec.
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.