Web App Testing Using Knockout.JS

Roberto Messora

Chapter No.1
"Web Application Testing in the Modern Web"
In this package, you will find:
The author’s biography
A preview chapter from the book, Chapter no.1 "Web Application Testing in the Modern Web"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Roberto Messora is a product manager and software architect for geospatial IT solutions based on ESRI, Microsoft .NET, and HTML5 technology stacks. In the last few years, working as a specialist at Value Lab S.p.a. (http://www.valuelab.it/en/), an innovative management consulting and IT solutions company specializing in marketing, sales, and retailing, he has acquired a deep knowledge in Location Analytics and Geomarketing, delivering web applications and tools for a wide range of industries.
He also believes in an effective and collaborative team environment, adopting modern ALM techniques and tools.

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
I'd like to thank my wife, Adelia, and my son, Alessandro, because sometimes they don't understand very well why I spend so much time on technology and software, but they still love and support me in everything I do. I love you too. I'd like to thank my parents because a solid house needs firm foundations.

I also want to thank everyone at UGIDotNET, the biggest Italian Microsoft .NET user group, in particular Andrea Saltarello: you are an awesome developer and a remarkable friend. Isaac Newton once said: "If I have seen further, it is by standing upon the shoulders of giants."

Finally, I'd like to thank everyone involved in the production of this book, especially all my editors at Packt Publishing: Subho Gupta, Arun Nadar, and Tanvi Bhatt. Also, I really want to thank my reviewers for every suggestion that made this book better. This is my first effort on a real book, in a language that is not my mother tongue, and they held my hands from the beginning of this journey.

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
Web App Testing Using Knockout.JS

This book is a starting point to meet the most important concepts and frameworks involved in modern web application production.

Delivering a web application nowadays consists of building a complete development environment, starting with a proper design and its testing, and ending with an automated build process.

Every chapter sequentially introduces a library or a tool as a specific building block of the entire environment, proposing concrete examples and development considerations: use a presentation framework with Knockout.JS, unit test in BDD style with Jasmine, design and implement a proper development strategy to achieve good unit testing coverage, and install and configure a build environment based on task automation.

This book shows a solid direction to follow when developing a web application and allows its readers to collect new hints and ideas to improve their own design strategies and delivery process.

What This Book Covers

Chapter 1, Web Application Testing in the Modern Web, is an introduction to modern web development, showing how important the testing of increasingly complex applications is. In particular, it shows how the classic client-side development based on jQuery is no longer the best solution when unit testing becomes one of the pillars of the implementation strategy. Also, it suggests that a new design approach is needed, based on a JavaScript MV presentation design pattern.

Chapter 2, The Knockout.JS UI Framework Explained, covers the most important features of Knockout.JS, which is the most well-known JavaScript presentation pattern based on Model-View-ViewModel. It also shows how to effectively use this library in a web application and why this is a good choice in terms of unit testing and application design.

Chapter 3, The Jasmine Unit Testing Framework Explained, covers the most important features of Jasmine, a very well-known JavaScript unit testing framework, based on the Behavior Driven Development approach.

Chapter 4, Unit Testing Strategies, is the core chapter of the book. It shows how to strategically deal with the development of a JavaScript web application when unit testing plays a major role. It suggests some important design principles and approaches when
developing a sample web application that uses Knockout.JS as the presentation framework and Jasmine for unit testing.

Chapter 5, Setting Up an Effective Testing Environment, shows how to set up an automated testing environment based on Gulp.JS and Karma; the first is a Node.JS module that acts as a task automation system, and the second is a test runner system. It covers the steps needed to install, configure, and execute all of the system components, including unit testing reports and code coverage. It also shows how to directly test the user interface using Phantom.JS and its page automation capabilities.
Web Application Testing in the Modern Web

Every software development book tries to explain some important technical aspects of a particular framework, technology, or methodology. This book is not different from this approach, but I'd like that everyone reading these chapters will try not only to think of the technicalities, but also to keep in mind the big picture.

Yes, I'd like it very much that we, that is everyone involved in software development, really start to think about the broader scenario, at least in general terms, as the most important objective of our work.

Every technique we learn, every methodology, analysis, and nonfunctional requirement, should be treated and chosen as a brick to build a comfortable house, for a variety of reasons. Concepts such as the end users, stakeholders, project economics, and so on should be handled as primary elements when deciding to adopt a particular technology.

Throughout this book, you will learn something very important in delivering modern web applications; I hope all of these notions will be used to improve web development awareness and understanding, not personal delight.

In this chapter, you will learn:

- Why software testing should be an essential habit
- Why the classic jQuery-style development is not the best choice, unless you are developing a general purpose plugin
- Why we should consider presentation design patterns as the cornerstone of the client-side web development

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
Software testing

One of the most debated topics in software development is testing. There are a lot of different opinions about many aspects of software testing, from its role in the design process, to the execution frequency and codebase coverage. For example, in most of the agile methodologies, unit testing plays a primary role, but not all projects can follow a strict test-first process. We have to find the best trade-off between the development process principles and the stakeholder’s environment.

Not every project is treated in the same way when we talk about software testing. From time to time, there are projects with a very low number of tests; not every part of the code base is covered by any kind of verification.

It happens, this is the real world, and it will continue to happen. However, this is not a healthy way to proceed. Basically, software testing is the proper way to prove that our solution:

• Works as expected
• Meets functional and nonfunctional requirements
• Is free of defects

There are contracts that consider that a project has concluded only in the presence of the appropriate documentation showing the correct system operation. To reach this objective, we need a solid testing strategy; this means that we need to carefully choose the following:

• The right testing levels (unit, integration, acceptance, and so on)
• The right testing types (regression, acceptance, usability, security, and so on)
• The right testing process (test first, waterfall, top-down, and so on)
• The right testing tools (frameworks, third-party products, dedicated environments, and so on)
• The right testing outputs/reports, depending on who will evaluate them

Every single topic in these bullet points carries years of professional experience; it’s not easy and immediate to become an expert. I also have to say that not every project covers all the enlisted items because of budget, time, or team habits.

On the other hand, there are two main reasons that suggest introducing at least a basic testing coverage from the bottom:

• Better design
• To avoid regression errors

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
This is a minimal scenario, but it's important to define a starting point, and I think this is the best place to begin a journey that will lead to a new way to consider software development.

This starting point is called **unit testing**, the first level of software testing; it works directly on the code and is usually the easiest to set up and learn.

This kind of testing should never be absent in any software solution. There are plenty of frameworks and libraries for the most important programming languages and **Integrated Development Environments (IDEs)**; not writing tests is just a matter of will.

This book talks about web application testing, but it's not intended to be an exhaustive reference for software testing; more of a short manual, so we need to focus on unit testing. However, it's very important to understand that you should consider this book in a broader context, a piece in the jigsaw.

In particular, this is not a book that talks about **Test Driven Development (TDD)**. TDD is a design methodology that leverages unit testing as the primary working tool. It's quite common to use TDD and unit testing as synonyms, but this is not correct. You cannot practice TDD without unit testing, but you can write unit tests without TDD and improve your design anyway.

In this book, I'd like to show what you need to do to set up a proper testing environment for a web solution, not teach you how to design your applications. Software design is a question of professional experience and involves a lot of general development aspects not specifically related to web development. On the other hand, what you will learn in this book is strictly related to the web application scenario, because we will talk about technical details, frameworks, and libraries that you can only find in web development.

By the way, you are welcome to apply TDD in web development using what you find in these chapters.

### The modern Web

The Web has changed a lot since HTML5 made its appearance. We are witnessing a gradual shift from a classic full server-side web development to a new architectural asset that moves much of the application logic to the client side.

The general objective is to deliver **rich internet applications (RIA)** with a desktop-like user experience. Think about web applications such as Gmail or Facebook. If you maximize your browser, they look like full desktop applications in terms of usability, UI effects, responsiveness, and richness.
Usually, solution architecture is quite simple, a rich and complex user interface on the client side, some application services on the server side. The technology scenario is also very straightforward. On the client side, HTML5, JavaScript, and CSS3 are the emerging options. They are replacing older plugin technologies such as Adobe Flash and Microsoft Silverlight. On the server side, HTTP RESTful services are widely adopted because of their implementation simplicity and standardization across a large variety of programming languages.

This is the main technology trend. Maybe it's not the best technological choice, but it has many advantages in terms of simplicity, integration, and user device penetration.

Widespread device diffusion is undoubtedly one of the most important key success factors of the magic trio, HTML5, JavaScript, and CSS3. In fact, we can assert that almost every device with a modern web browser (PC, tablet, or smartphone) is able to access and deliver an HTML5 application to the user.

In other words, this is why we are here. There are millions of devices out there that allow end users to access our web applications, and we need to assure a reliable interaction for them. Think about home banking web sites. If you access your bank account, you won't feel very comfortable if a money transfer fails because of an application defect. Maybe because developers forgot a critical data validation or didn't properly handle a remote service error, you don't know if your transfer has been accepted or not.

Web applications are growing in complexity, not just because of modern web technologies, but also because of marketing strategies, product placements, or simple market competition. In this scenario, tightly related to the visibility and reputation of a company, we cannot approve a delivery plan that doesn't provide some kind of testing strategy.

Once we establish that testing is a pillar of our solution, we need to understand which is the best way to proceed in terms of software architecture and development. In this regard, it's very important to determine the very basic design principles that allow a proper approach to unit testing. In fact, even though HTML5 is a recent achievement, HTML and JavaScript are technologies that have been in use for quite some time.

The problem here is that many developers tend to approach modern web development in the same old way. This is a grave mistake because back in time, client-side JavaScript development was a underrated and mostly confined to simple UI graphic management.

For More Information:  
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
Client-side development is historically driven by libraries such as Prototype, jQuery, and Dojo, whose primary feature is DOM (HTML Document Object Model or HTML markup) management. They can work in small web applications, but as soon as these grow in complexity, the codebase starts to become unmanageable and unmaintainable. We can't think that we can continue to develop JavaScript in the same way we did 10 years ago. In those days, we only had to apply some of the UI transformations dynamically. Today, we have to deliver full working applications.

We need a better design, but most of all we need to reconsider the client-side JavaScript development and apply the advanced design patterns and principles.

Coming back to the context of this book, a new development approach is essential if one of our objectives is a testable codebase. In fact, once we decide that we want to test our solution, we need to write testable code.

Does this sound silly? Well… not at all. We can't simply say that we want to test our code. We must ensure that it is possible. Not all code is testable.

Roy Osherove in his book, The Art of Unit Testing, gives the following definition:

"A unit test is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed. A unit is a method or function."

Martin Fowler also writes about unit testing and the concept of isolation (http://martinfowler.com/bliki/UnitTest.html):

"A more important distinction is whether the unit you're testing should be isolated from its collaborators. Imagine you're testing an order class's price method. The price method needs to invoke some functions on the product and customer classes. If you follow the principle of collaborator isolation you don't want to use the real product or customer classes here, because a fault in the customer class would cause the order class's tests to fail. Instead you use TestDoubles for the collaborators."

These definitions of unit testing talk about code that verifies other code isolating individual parts of a program. It's clear that we need to provide at least these basic characteristics to our code if we want to test it. In this book, we are going to suggest some useful design principles, but we won't focus on software development methodologies (for example, Extreme Programming, Waterfall, Agile, and others).
Escaping from the jQuery-style development

The World Wide Web is a fairly simple environment; all the content available is accessible using a web browser. Also, web browsers are conceptually simple applications. They ask for a web resource, a server sends that resource, and they display what's inside the received data. A web resource is nothing else than a series of files that represents different types of content:

- **User interface structures**: HTML markup
- **User interface graphic styles**: CSS
- **User interface programs and executables**: JavaScript
- **Data**: JSON or XML

Every browser supports these four types of content even if there are some interpretation differences (global standardization has not yet been reached).

We can say that JavaScript is the programming language of the Web, but its native DOM API is somewhat rudimentary. We have to write a lot of code to manage and transform HTML markup to bring the UI to life with some dynamic user interaction. Also, full standardization does not mean that the same code will work differently (or work at all) in different browsers.

Over the past few years, developers decided to resolve this situation; JavaScript libraries such as Prototype, jQuery, and Dojo have come to light.

jQuery is one of the most known open source JavaScript libraries, published in 2006 for the first time. Its huge success is mainly due to the following reasons:

- A simple and detailed API that allows us to manage HTML DOM elements
- Cross-browser support
- Simple and effective extensibility

Since its appearance, it has been used by thousands of developers as the foundation library. A large amount of JavaScript code all around the world has been built with jQuery in mind. A jQuery ecosystem grew up very quickly, and nowadays there are plenty of jQuery plugins that implement virtually everything related to web development.

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
A typical jQuery-based web application is something like the following sample page:

```html
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>jQuery Sample</title>
  <style type="text/css">
    ... 
  </style>
</head>
<body>
  <h4>jQuery Sample</h4>
  <form id="inputForm">
    <label for="firstName">Name:&nbsp;</label>
    <br>
    <input type="text" id="firstName" name="firstName">
    <br>
    <label for="lastName">Last Name:&nbsp;</label>
    <br>
    <input type="text" id="lastName" name="lastName">
    <br>
    <fieldset>
      <legend>Nationality:</legend>
      <input type="radio" name="nat" value="it" checked>Italian
      <br>
      <input type="radio" name="nat" value="foreign">Foreign
      <br>
      <select id="foreignNat" disabled="disabled">
        <option value="">Select an item...</option>
        <option value="us">American</option>
        <option value="uk">British</option>
        <option value="fr">French</option>
        <option value="other">Other</option>
      </select>
    </fieldset>
    <input type="submit" value="Submit">
    <br>
    <p id="resultMessage"></p>
  </form>
<script type="text/JavaScript" src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>
<script type="text/JavaScript">
  //JavaScript Code
</script>
</body>
</html>
```

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
This is a simple HTML page with a single form and bunch of inputs:

If we want to validate this form before submission, we can add the following code in the script tag:

```javascript
function showMessage (message, isError) {
    jQuery("#resultMessage").text(message);
    if (isError) {
        jQuery("#resultMessage").css("color", "red");
    } else {
        jQuery("#resultMessage").css("color", "green");
    }
    jQuery("#resultMessage").css("visibility", "visible");
}
```

For More Information:
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
function hideMessage () {
    jQuery("#resultMessage").css("visibility", "hidden");
}

hideMessage();

jQuery(document).ready(function () {
    jQuery("#inputForm input:radio").on("click", function () {
        if (jQuery(this).val() === "it") {
            jQuery("#foreignNat option").removeAttr("selected");
            jQuery("#foreignNat").attr("disabled", "disabled");
        } else {
            jQuery("#foreignNat").removeAttr("disabled");
        }
    });
    jQuery("form").submit(function (e) {
        e.preventDefault();
        hideMessage();
        jQuery("#resultMessage").text("Please fill First Name");
        if (!jQuery("#firstName").val()) {
            showMessage("Please fill First Name", true);
            return;
        }
        if (!jQuery("#lastName").val()) {
            showMessage("Please fill Last Name", true);
            return;
        }
        if (!jQuery("input[name=nat]:checked").val()) {
            showMessage("Please select Nationality", true);
            return;
        }
        if ((jQuery("input[name=nat]:checked").val() === "foreign") &&
            (!jQuery("#foreignNat option:selected").val())) {
            showMessage("Please select Foreign Nationality", true);
            return;
        }
        //form submission, for example with an AJAX call
        showMessage("Form submitted", false);
    });
});
The preceding code is a typical example of jQuery-style programming. Nearly every line contains a jQuery call, from pure tag content management to event handling. It works and is also a simple API to learn; an average developer becomes productive in a very short period of time.

In the preceding code, we defined two helper functions (showMessage and hideMessage) to show and hide the alert message. Then, when the page is fully loaded (jQuery(document).ready(...)), we defined two event handler functions: one for the radio button click (jQuery("#inputForm input:radio").on("click", ...) and the other for the form submission (jQuery("form").submit(...)).

In the first one, we activate or disable the drop-down menu, whether or not the Foreign radio option is checked:
In the second event handler function, we make some input validation and proceed with form submission only if all the inputs are properly filled, providing a specific status message.

The preceding example is a simple one: there are a few HTML tags, the UI logic is quite simple (just a message that appears/disappears and drop-down menu that activates/deactivates), and the validation is pretty straightforward.

Yet, despite its simplicity, all of the code that we just wrote is virtually untestable. There are two main reasons:

- User interface items are tightly coupled with the user interface logic
- The user interface logic spans inside the event handler callback functions

The real problem here is that everything passes through a jQuery reference, that is, a `jQuery("something")` call. This means that we always need a live reference of the HTML page, otherwise those calls fail, and this is also true for a unit test case. We can't think of testing a piece of user interface logic running an entire web application!
Consider a real-case scenario, not this super simple example of a complex e-commerce website, and we need to test a specific page. We need to reach this page first (maybe we also need to enter some user credentials because it's a private page), provide some input data, and then start our test case. Think of repeating this journey as many times as different input data combinations are possible, or try to automate this process in someway. A real nightmare!

Worse than this, you have to repeat the entire process if you simply change a single HTML tag ID and nothing else. The UI logic is unchanged, but we need to assure that no jQuery tag reference breaks if we change something in the HTML markup. This is really too much work to do. We will soon realize that we spend more time running unit tests than developing the application.

On the other hand, large jQuery applications tend to be monolithic because jQuery itself allows the callback function nesting very easily and doesn't really promote any particular design strategy. The result is often a spaghetti code.

jQuery is a good option if you want to develop a specific custom plugin. We will also continue to use this library for pure user interface effects and animations, but we need something different to maintain a large web application's logic.

**Presentation design patterns**

In the previous section, you learned that a development approach based on jQuery is not a good idea for a large web application.

To move a step forward, we need to decide what's the best option in terms of testable code. The main topic here is the application design; in other words, how we can build our codebase following a general guideline with testability in mind.

In software engineering, there's nothing better than not reinventing the wheel; we can rely on a safe and reliable resource: design patterns. Wikipedia provides a good definition for the term design pattern (http://en.wikipedia.org/wiki/Software_design_pattern):

"In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design.

A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations.

Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system."

For More Information:  
www.packtpub.com/web-development/web-app-testing-using-knockoutjs
There are tons of specific design patterns, but we also need something related to the presentation layer because this is where a JavaScript web application belongs.

The most important aspect in terms of design and maintainability of a JavaScript web application is a clear separation between the user interface (basically, the HTML markup) and the presentation logic (the JavaScript code that turns a web page dynamic and responsive to user interaction). This is what you learned digging into a typical jQuery web application.

According to MSDN's *patterns & practices*, there are other important aspects involved in the presentation design patterns (http://msdn.microsoft.com/en-us/library/ff647343.aspx):

- Document simple mechanisms that work
- Provide a common vocabulary and taxonomy for developers and architects
- Enable solutions to be described concisely as combinations of patterns
- Enable the reuse of architecture, design, and implementation decisions
- You have to support user interactions of increasing complexity, involving complex relationships between forms and pages in the user interface
- Existing business processes change and you have to present new or modified functionalities to your users
- You have to port your application to other platforms or make the application accessible to additional client types (such as mobile devices)

At this point, we need to identify an effective implementation of a presentation design pattern and use it in our web applications. In this regard, I have to admit that the JavaScript community has made an extraordinary job in the last two years. So far, there are tons of frameworks and libraries that implement a particular presentation design pattern.

We only need to choose which framework fits our needs, for example, we can start by taking a look at the MyTodo MVC website (http://todomvc.com/). This is an open source project that shows how to build the same web application using a different library each time.

In this book, we will use **Knockout.JS**, one of the most popular libraries that implements the **Model-View-ViewModel** design pattern.
Most of these libraries implement a so-called MV* design pattern (Knockout.JS does this too). MV* means that every design pattern belongs to a broader family with a common root, which is Model-View-Controller. The MVC pattern is one of the oldest and most enduring architectural design patterns. Originally designed by Trygve Reenskaug, working on Smalltalk-80 back in 1979, it has been heavily refactored since then.

Basically, the MVC pattern enforces the isolation of business data (Models) from user interfaces (Views), with a third component (Controllers) managing the logic and user input. It can be described as (Addy Osmani, Learning JavaScript Design Patterns, http://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailmvc):

- A Model representing domain-specific data and ignorant of the user interface (Views and Controllers). When a model is changed, it will inform its observers.
- A View representing the current state of a Model. The Observer pattern was used to let the View know whenever the Model was updated or modified.
- Presentation was taken care of by the View, but there wasn’t just a single View and Controller—a View-Controller pair was required for each section or element being displayed on the screen.
- The Controller's role in this pair handles user interaction (such as key-presses and actions, for example, clicks), making decisions for the View.

This general definition has slightly changed over the years, not only to adapt its implementation to different technologies and programming languages, but also because changes have been made to the Controller part. Model-View-Presenter and Model-View-ViewModel are the most well known alternatives to the MVC pattern.

The MV* presentation design patterns are a valid answer to our need; an architectural design guideline that promotes separation of concerns and isolation, the two most important factors needed for software testing. In this way, we can test the models, views, and third actor whatever it is (a Controller, Presenter, ViewModel, and so on) separately.

On the other hand, adopting a presentation design pattern doesn’t mean that we cease to use jQuery. jQuery is a great library and we will continue to add its reference to our pages, but we will also integrate its use wisely in a better design context.
Summary

This chapter gives a significant vision about software testing in the contemporary web era, starting with the most important consideration; it's unimaginable to think of a web application that is not supported by a solid testing strategy.

You learned that modern web development can't follow the same old paradigms such as using the well known jQuery library as the application cornerstone. Complex web applications tend to become unmanageable and untestable without a good architectural design. Every framework or library should be used in the correct context. jQuery has advantages in developing a plugin and pure user interface customizations, but it's not the best option if adopted as an application logic framework.

We finally saw that if we want to satisfy both maintainability and testability, we need to adopt a presentation design pattern. In the JavaScript community, the emerging presentation frameworks and libraries are a part of the MV* family and they promote separation of concerns and isolation between their components. In particular, Knockout.JS is one of the most well-known libraries and implements a specific pattern variation, which is Model-View-ViewModel.

In the next chapter, you will learn more about Knockout.JS and its features.

For More Information:

www.packtpub.com/web-development/web-app-testing-using-knockoutjs
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

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