Completing the Picture

In his must-read book, Working Effectively with Legacy Code, Michael Feathers states:

Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse.

Those are strong words. However, we completely agree with him. What he calls the "Cover [with tests] and Modify" approach is demonstratably more efficient than the "Edit and Pray [because there are no automated tests]" approach. And although most of us know this, either through our formal education or through the "school of hard knocks", oftentimes we find ourselves writing what Mr. Feathers calls "bad code". The reasons vary, but typically it's because writing testable code can be difficult, especially when developing on top of certain frameworks.

The good news, in this regard, is that ASP.NET Web API was developed with testability in mind. In this chapter we will demonstrate how to achieve high levels of code coverage relatively easily.

Finally, based on feedback from the previous edition of the book, we will demonstrate how to consume the task-management service using a simple ASP.NET MVC Web application.

We're in the homestretch now, so let's finish strong!

# Testing the API

Our usual approach to automated testing employs a mix of integration tests and unit tests, with a bias towards unit tests. This is because though integration tests are needed to ensure everything works together, unit tests typically provide much less "friction" when trying to achieve high levels of code coverage (e.g., unit tests don't require access to a database).

In this section we will test the Get Tasks operation. We have chosen Get Tasks because it represents a superset of most of the other operations; i.e., it involves paging, database access, type mapping, hypermedia links, etc. We'll begin by putting unit tests in place, and then we'll go end-to-end with an integration test. A familiarity with NUnit and Moq would be helpful at this point; however, you will still be able to benefit by following along with the test implementation, regardless of your experience with these frameworks.

## Unit Testing

The first thing we need to do is add the aforementioned 3rd party dependencies. With the solution open, run the following commands in the Package Manager Console to install the testing and mocking frameworks that we introduced in Chapter 3, respectively:

install-package NUnit WebApi2Book.Web.Api.Tests

install-package Moq WebApi2Book.Web.Api.Tests

Next, add

## Integration Testing

# Going Live!

TODO - Jamie's section on the UI, CORS, CSRF…

# Summary