THE EXPERT'S VOICE® IN .NET

Practical ASP.NET Web API

MASTER THE ASP.NET WEB API FRAMEWORK IN A PRACTICAL, HANDS-ON, WAY

Badrinarayanan Lakshmiraghavan

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

"I hear...I forget, I see...and I remember, I do...and I understand"

—Confucius

The Hypertext Transfer Protocol (HTTP) is the application-level protocol that powers the World Wide Web. One of the greatest characteristics of HTTP is that it finds support in multiple platforms. HTTP is the lowest common denominator of many platforms and devices. Hence, the primary benefit of creating an HTTP-based service is reachability. A broad range of clients in disparate platforms can consume your HTTP services.

ASP.NET Web API is a framework from Microsoft for building HTTP services. It is not the only possible means for building HTTP services in the .NET technology stack; there is Windows Communication Foundation (WCF) as well, but the ASP.NET Web API framework embraces HTTP instead of fighting against it or abstracting it away. ASP.NET Web API enables you to create HTTP services through the powerful ASP.NET MVC programming model of preferring convention over configuration, which is familiar to many .NET web developers. Some of the best features from ASP.NET MVC, such as routing, model binding, and validation, are all part of ASP.NET Web API as well. ASP.NET Web API also lends itself well to unit testing, in a similar way toASP.NET MVC.

This book, Practical *ASP.NET Web API*, is a practical guide that will help you master the basics of the great ASP.NET Web API framework in a hands-on way. It takes a code-centric approach that will help you grasp the concepts by seeing them in action as you code, run, and debug the projects that you create as you follow the exercises of a chapter.

Though the main focus of the book is the practice, which is the 'how' part of ASP.NET Web API framework development, the 'what' and 'why' parts are implicitly covered to the extent needed for you to understand and appreciate the underlying theoretical concepts demonstrated by the practical code, as you work through the various scenarios. You will see a lot of code, covering all the practical and basic scenarios that are commonly encountered by developers. The recommended approach that will provide the maximum benefit is to follow this book's exercises in sequence and code-along. Although it is a bit of work, I recommend you manually type the code instead of copying and pasting it from the book's download files into the Visual Studio classes you work on. This will help you grasp what you are trying to do, as you work through an exercise. However, if having the completed source code by your side will be of help, you can find the code for the examples shown in this book on the Apress web site, `www.apress.com`. A link can be found on the book's information page under the Source Code/Downloads tab.

If you are looking for a book to just read through and gain an understanding of the ASP.NET Web API framework by simply looking at code listings, this is mostly not your book. While you will see lots of code, this is not a recipe book. Though you will find the code listings in the book useful and relevant for many of the scenarios you face day-to-day, the intention of this book is not to provide you ready-made code that you can copy and paste into the code you are working on in a real-life project and forge ahead. The objective instead is to provide you the hands-on experience of learning the basics of the ASP.NET Web API framework. In short, this book follows the proverb quoted in the epigraph—do, and you will understand.

What You'll Learn

- The basics of HTTP services and debugging through Fiddler.
- Request binding and validation.
- Response formatting and customization to suit client preferences.
- Managing the controller dependencies and unit testing.
- Hosting and security fundamentals.
- Consuming HTTP services from various clients.
- Building a performant web API.

How This Book is Organized

Practical *ASP.NET Web API* is organized into twelve chapters built around hands-on exercises. Each exercise builds on the previous one and for this reason, I highly recommend not only reading the chapters in order but also following the exercises within a chapter in the order presented. You'll find the following chapters in this book.

Chapter 1: Building a Basic Web API

We start off by understanding the differences in building HTTP services using Windows Communication Foundation (WCF) versus ASP.NET Web API at a high level and move on to building our first service, which exposes an in-memory collection over HTTP. We then look at overriding the default behavior of the ASP.NET Web API framework in selecting the action methods based on the HTTP method and finish off the chapter by creating a create-read-update-delete service that plays by the rules of HTTP.

Chapter 2: Debugging and Tracing

The ability to view HTTP traffic, which consists of the request message sent by the client and the response message sent by ASP.NET Web API in response to the request, and the ability to hand-craft requests and submit the same to ASP.NET Web API to view the corresponding response are fundamental requirements for building HTTP services. This chapter covers Fiddler, a great tool for HTTP debugging, and the web browsers' built-in tools to capture and inspect the HTTP traffic. This chapter also covers the tracing feature that comes with the ASP.NET Web API framework.

Chapter 3: Media-Type Formatting CLR Objects

This chapter introduces you to the concept of formatting, which is introduced in the ASP.NET Web API framework. You will understand how the process of content negotiation (conneg) works and learn to override and extend it. You will create media type mappings through a query string and request header, a custom media type mapping, and a custom media formatter, and you'll learn to extend the out-of-box JSON media formatter. Finally, you'll look at controlling what and how members of a type get serialized into HTTP response.

Chapter 4: Customizing Response

Content negotiation is not just about choosing the media type for the resource representation in the response. It is also about the language, character set, and encoding. In Chapter 3, content negotiation is covered from the media type perspective. This chapter explores content negotiation from the perspective of language, character set, and content encoding.

Chapter 5: Binding an HTTP Request into CLR Objects

This chapter introduces the concept of binding, which is borrowed from the ASP.NET MVC framework. Binding in ASP.NET Web API is much broader, with media type formatters also having a role to play. You will learn the three types of binding: model binding, formatter binding, and parameter binding; and you'll learn how to extend the framework by creating custom value providers, custom model binders, custom parameter binders, and custom media-formatters.

Chapter 6: Validating Requests

This chapter covers model validation, a process that is part of model binding, by which ASP.NET Web API runs the validation rules you set against the properties of your model classes. You will use the out-of-box data annotations to enforce the validity of the incoming request and handle the errors raised by model validation. You will also extend the out-of-box validation attribute, create your own validation attribute, and create a validatable object.

Chapter 7: Managing Controller Dependencies

This chapter covers the techniques related to managing one of the most common dependencies an ASP.NET Web API controller takes, which is the dependency on the classes related to persistence infrastructure such as a database. You start off building a controller that depends on Entity Framework and move on to invert the dependencies using the interfaces part of Entity Framework; this is followed by applying the repository pattern and generalizing that pattern into a generic repository. You will also look at mapping domain objects to data transfer objects (DTO) using AutoMapper, injecting dependencies using StructureMap, and writing automated unit tests against the controller by using RhinoMocks as the mocking framework.

Chapter 8: Extending the Pipeline

ASP.NET Web API is a framework. You don't call the framework code but it calls your code, in line with the Hollywood principle. The most fundamental lever that you use to harness the power of ASP.NET Web API framework is the controller, the `ApiController` subclass that you write. In addition, the ASP.NET Web API framework has various points of extensibility built in, for us to hook our code in and extend the processing. This chapter covers the extensibility points of message handlers, filters, and controller selectors.

Chapter 9: Hosting ASP.NET Web API

Though ASP.NET Web API includes ASP.NET in its name, it is not tied to the ASP.NET infrastructure. In fact, ASP.NET Web API is host-independent. This chapter covers the three ways you can host your HTTP services built using ASP.NET Web API: (1) using the ASP.NET infrastructure backed by Internet Information Services (IIS), called web hosting, (2) using any Windows process such as a console application, called self-hosting, and (3) connecting client to the web API runtime, without hitting the network, called in-memory hosting and used mainly for testing purposes.

Chapter 10: Securing ASP.NET Web API

Authentication and authorization are the fundamental building blocks to secure any application, including ASP.NET Web API-powered HTTP services. This chapter covers HTTP basic authentication as an example for implementing the direct authentication pattern and a client obtaining a JSON Web Token (JWT) from an issuing authority and presenting the same to ASP.NET Web API as an example for brokered authentication pattern. This chapter also covers authorization based on roles implemented using `Authorize` filter.

Chapter 11: Consuming ASP.NET Web API

One of the greatest benefits of building HTTP services is the reachability. A broad range of clients in disparate platforms can consume your HTTP service, leveraging the support HTTP enjoys across the platforms and devices. This chapter covers the topic of the client applications consuming your ASP.NET Web API. The coverage is limited to two .NET clients: a console application and a Windows Presentation Foundation (WPF) application, and a JavaScript client running in the context of a browser.

Chapter 12: Building a Performant Web API

Performance, an indication of the responsiveness of an application, can be measured in terms of latency or throughput. Latency is the time taken by an application to respond to an event, while throughput is the number of events that take place in a specified duration of time. Another quality attribute that is often used interchangeably is scalability, which is the ability of an application to handle increased usage load without any (or appreciable) degradation in performance. The topics of performance and scalability are vast and hence this chapter focuses on a few select areas in ASP.NET Web API, namely asynchronous action methods, pushing real-time updates to the client, and web caching.

What You Need to Use This Book

All the code listing and the samples in this book are developed using Visual Studio 2012 Ultimate Edition, targeting the .NET framework 4.5 in Windows 7 and hence Visual Studio 2012 is a must to use this book. Since ASP.NET Web API is a part of ASP.NET MVC 4.0 and it ships with Visual Studio 2012, you will not need any separate installs to get the ASP.NET Web API framework.

For the exercises that involve creating automated unit tests, I used Visual Studio Unit Testing Framework. To work through those exercises, you will need a minimum of the professional edition of Visual Studio to create and run the tests but the ultimate edition is recommended.

In addition to Visual Studio, you will need IIS for web-hosting your web API and Microsoft SQL Server 2012, either the Express edition or preferably the Developer edition, to be used as the persistence store. You will also need the browsers: mostly Internet Explorer 9.0 and Google Chrome in some specific cases. You'll also need the HTTP debugging tool Fiddler (`http://fiddler2.com/`). For the exercises that require external .NET assemblies, you can use NuGet from Codeplex (`http://nuget.codeplex.com/`) to pull those packages into your project. For Chapter 12 on performance, in order to simulate some load, you will need Apache Bench (ab.exe), which is part of Apache HTTP Server.

Who This Book Is For

The book is for every .NET developer who wants to gain a solid and a practical hands-on understanding of the basics of the ASP.NET Web API framework. A good working knowledge of C# and the .NET framework 4.5, familiarity with Visual Studio 2012 are the only pre-requisites to benefit from this book, though a basic knowledge of the ASP.NET MVC framework and the HTTP protocol will be helpful.

CHAPTER 1

Building a Basic Web API

A web API is just an application programming interface (API) over the web (that is, HTTP). When the resources of an application can be manipulated over HTTP using the standard HTTP methods of GET, POST, PUT, and DELETE, you can say that the application supports a web API for other applications to use. Because HTTP is platform-agnostic, HTTP services can be consumed by disparate devices across different platforms.

A central concept of HTTP services is the existence of resources that can be identified through a uniform resource identifier (URI). If you equate resources to nouns, then actions on a resource can be equated to verbs and are represented by the HTTP methods such as GET, POST, PUT, and DELETE. For an application that deals with the employees of an organization, each employee is a resource the application deals with.

Let us see how an employee's details can be retrieved with an HTTP service. The URI is `http://server/hrapp/employees/12345`. It includes the employee ID and serves as an identifier to the resource, which is an employee in this case. Actions on this resource are accomplished through the HTTP verbs. To get the details of an employee, you will perform an HTTP GET on the URI `http://server/hrapp/employees/12345`. To update this employee, the request will be an HTTP PUT on the same URI. Similarly, to delete this employee, the request will be an HTTP DELETE request, again on the same URI. To create a new employee, the request will be an HTTP POST to the URI without any identifier (`http://server/hrapp/employees`).

In the case of POST and PUT, the service must be passed the employee data or the resource representation. It is typically XML or JSON that is sent as the HTTP request message body. An HTTP service sends responses in XML or JSON, similar to the request. For example, a GET to `http://server/hrapp/employees/12345` results in a response containing JSON representing the employee with an ID of 12345.

HTTP service responds with the HTTP status code indicating success or failure. For example, if the employee with identifier 12345 does not exist, the HTTP status code of `404 - Not found` will be returned. If the request is successful, the HTTP status code of `200 - OK` will be returned.

The ASP.NET Web API framework enables you to create HTTP-based services through the powerful ASP.NET MVC programming model familiar to many developers. So, we have the URI `http://server/hrapp/employees/12345`, and a client issues a GET. To respond to this request, we need to write code somewhere that retrieves the employee details for 12345. Obviously, that code has to be in some method in some C# class. This is where the concept of routing comes into play.

The class in this case typically will be one that derives from the `ApiController` class, part of the ASP.NET Web API framework. All you need to do is to create a subclass of `ApiController`, say `EmployeesController`, with a method `Get(int id)`. The ASP.NET Web API framework will then route all the GET requests to this method and pass the employee ID in the URI as the parameter. Inside the method, you can write your code to retrieve the employee details and just return an object of type `Employee`. On the way out, the ASP.NET Web API framework will handle serialization of the employee object to JSON or XML. The web API is capable of content negotiation: A request can come in along with the choices of the response representation, as preferred by the client. The web API will do its best to send the response in the format requested.

In the case of requests with a message payload such as POST, the method you will need to define will be `Post(Employee employee)` with a parameter of type `Employee`. The ASP.NET Web API framework will deserialize the request (XML or JSON) into the `Employee` parameter object for you to use inside the method. The web API dispatches a request to an action method based on HTTP verbs.

ASP.NET MVC 4 ships as part of Visual Studio 2012 and as an add-on for Visual Studio 2010 SP1. ASP.NET Web API is a part of MVC 4.0. There is a new project template called WebAPI available to create web API projects. You can have both API controllers and MVC controllers in the same project.

1.1 Choosing ASP.NET Web API or WCF

If you have worked with the .NET Framework for any amount of time, you must have encountered the term WCF (Windows Communication Foundation), the one-stop framework for all service development needs in the .NET Framework. Why the new framework of ASP.NET Web API then?

The short answer is that ASP.NET Web API is designed and built from the ground up with only one thing in mind—HTTP—whereas WCF was designed primarily with SOAP and WS-* in mind, and Representational State Transfer (REST) was retrofitted through the WCF REST Starter Kit. The programming model of ASP.NET Web API resembles ASP.NET MVC in being simple and convention-based, instead of requiring you to define interfaces, create implementation classes, and decorate them with several attributes. However, ASP.NET Web API is not supposed to supersede WCF.

It is important to understand the coexistence of WCF and ASP.NET Web API. WCF has been around for a while, and ASP.NET Web API is a new kid on the block, but that does not mean WCF is meant to be replaced by ASP.NET Web API. Both WCF and ASP.NET Web API have their own place in the big picture.

ASP.NET Web API is lightweight but cannot match the power and flexibility of WCF. If you have your service using HTTP as the transport and if you want to move over to some other transport, say TCP, or even support multiple transport mechanisms, WCF will be a better choice. WCF also has great support for WS-*.

However, when it comes to the client base, not all platforms support SOAP and WS-*. ASP.NET Web API–powered HTTP services can reach a broad range of clients including mobile devices. The bottom line: it is all about trade-offs, as is the case with any architecture.

Let's try to understand the differences in programming models by looking at a simple example: an employee service to get an employee of an organization, based on the employee ID. WCF code (see Listing 1-1) is voluminous, whereas ASP.NET Web API code (see Listing 1-2) is terse and gets the job done.

Listing 1-1. Getting an Employee the WCF Way

```
[ServiceContract]
public interface IEmployeeService
{
        [OperationContract]
        [WebGet(UriTemplate = "/Employees/{id}")]
        Employee GetEmployee(string id);
}

public class EmployeeService : IEmployeeService
{
        public Employee GetEmployee(string id)
        {
                return new Employee() { Id = id, Name = "John Q Human" };
        }
}
```

```
[DataContract]
public class Employee
{
         [DataMember]
        public int Id { get; set; }

         [DataMember]
        public string Name { get; set; }

        // other members
}
```

Listing 1-2. Getting an Employee the ASP.NET Web API Way

```
public class EmployeeController : ApiController
{
        public Employee Get(string id)
        {
                return new Employee() { Id = id, Name = "John Q Human" };
        }
}
```

A couple of things are worth mentioning here: First, the web API is exactly the same as a normal MVC controller except that the base class is `ApiController`. Features of MVC that developers like, such as binding and testability, which are typically achieved through injecting a repository, are all applicable to a web API as well.

If you are experienced with ASP.NET MVC, you may be wondering how different a web API is; after all, the MVC controller's action method can also return `JsonResult`. With `JsonResult` action methods, a verb is added to the URI (for example, `http://server/employees/get/1234`), thereby making it look more like RPC style than REST. Actions such as GET, POST, PUT, and DELETE are to be accomplished through HTTP methods rather than through anything in the URI or query string. ASP.NET Web API also has far superior features, such as content negotiation. ASP.NET MVC's support for `JsonResult` is only from the perspective of supporting AJAX calls from the JavaScript clients and is not comparable to ASP.NET Web API, a framework dedicated to building HTTP services.

The following are the scenarios where ASP.NET Web API as the back end really shines and brings the most value to the table:

- **Rich-client web applications:** ASP.NET Web API will be a good fit for rich-client web applications that heavily use AJAX to get to a business or data tier. The client application can be anything capable of understanding HTTP; it can be a Silverlight application, an Adobe Flash–based application, or a single-page application (SPA) built using JavaScript libraries such as JQuery, Knockout, and so on, to leverage the power of JavaScript and HTML5 features.
- **Native mobile and non-mobile applications:** ASP.NET Web API can be a back end for native applications running on mobile devices where SOAP is not supported. Because HTTP is a common denominator in all the platforms, even the native applications can use a .NET back-end application through the service façade of a web API. Also, native applications running on platforms other than Windows, such as a Cocoa app running on Mac, can use ASP.NET Web API as the back end.
- **Platform for Internet of Things (IOT):** IOT devices with Ethernet controllers or a Global System for Mobile Communications (GSM) modem, for example, can speak to ASP.NET Web API services through HTTP. A platform built on .NET can receive the data and do business. Not just IOT devices, but other HTTP-capable devices such as radio frequency ID (RFID) readers can communicate with ASP.NET Web API.