# Code Examples

|  |  |  |
| --- | --- | --- |
| Feature | Comments | Complete |
| Context-sensitive hyperlinks |  | X |
| Paging of results |  | X |
| partial updates |  | X |
| api versioning |  | X |
| legacy SOAP support |  | X |
| How to handle non-resource APIs using REST | TasksWorkflowController | X |
| How to best expose relationships between resources |  | X |
| entities vs web models | Guard against overposting. Using AutoMapper to translate. | X |
| IHttpActionResult | In global error handling, in POST for Task and User, and possibly in security masking. | X |
| async filter | Audits task re-activation. | X |
| input validation |  | X |
| attribute-based routing | Including prefixes, constraints, and coexistence with global, convention-based routing | X |
| content negotiation |  | X |
| CORS | cors vs jsonp (jsonp is hacky). enablecors/disablecors attributes |  |
| ASP.NET Identity | JAMIE!!! |  |
| Basic Auth |  | X |
| Bearer Auth |  |  |
| custom legacy auth | use route-specific handler |  |
| CSRF | Are we going to include this in code examples? JAMIE!!! |  |
| authorization attribute |  | X |
| require ssl attribute | optional |  |
| security masking of response message |  | X |
| Automatic lifetime management for database connections and transactions |  | X |
| NHibernate |  | X |
| tracing |  | X |
| global error handling |  | X |
| web page to illustrate how to consume the api | Shane may help with this |  |
| testing (unit, integration) |  |  |
| comments/documentation |  |  |

# Original Book Proposal to Apress

### Brief Book Description (two sentence summary --in simple, non-technical language):

Learn how to leverage the features and capabilities of the ASP.NET Web API to build a RESTful web service from start to finish. This book will first explain the REST architecture, and then build on that knowledge, the Web API, and a few other patterns, tricks, and techniques, to go from a blank slate to a fully functional, secure, and versioned REST service.

### Long Description (300-350 words) statement that defines the benefits offered to the reader:

The ASP.NET MVC Framework has always been a good platform on which to implement REST-based services, but the introduction of the ASP.NET Web API Framework raised the bar to a whole new level. Now in release version 2.1, the Web API Framework has evolved into a powerful and refreshingly usable platform. This concise book provides technical background and guidance that will enable you to best use the ASP.NET Web API 2 Framework to build world-class REST services.

As with the Web API 2 Framework itself, ASP.NET Web API 2: Building a REST Service from Start to Finish improves upon its predecessor, ASP.NET MVC4 and the Web API. New content in this edition includes

* New capabilities in Web API 2 (currently version 2.1).
* Swappable data storage. NHibernate, Entity Framework, and Dapper with Web API are featured.
* Support for partial updates, or PATCH.
* API versioning.
* Support for legacy SOAP-based operations.
* How to handle non-resource APIs using REST
* How to best expose relationships between resources
* JSON Web Tokens, CORS, CSRF, and OAUTH

Also, in response to feedback we received on the first edition, we removed or relegated to an appendix the non Web API content that was deemed to be superfluous or distracting.

So get ready for authors Jamie Kurtz and Brian Wortman to take you from zero to REST service hero in no time at all. No prior experience with ASP.NET Web API is required; all Web API-related concepts are introduced from basic principles and developed to the point where you can use them in a production system. A good working knowledge of C# and the .NET Framework are the only prerequisites to best benefit from this book.

Although some examples include ASP.NET MVC features, the emphasis is on the Web API and not MVC. We recommend Pro ASP.NET MVC 5 if you need help on topics relating to MVC and/or CSS, HTML, JavaScript, jQuery, the Razor view engine, HTML Helpers, model binding, etc.

### Why is this an important topic? How large is the market? Please cite any market statistics or other relevant sources.

RESTful services are growing in importance at an incredible rate right now, mostly in support of the variety of devices out there today. REST style services are being used to provide backends for nearly all mobile applications, as well as, the newer style of web sites being built - i.e. single-page apps.

### Target audience: Level and Technical Focus

Beginner to Intermediate; REST, .NET, C#

What The Reader Will Learn (in 5 or fewer bullet points):

* Introduction to the REST architecture
* How to design a REST API
* New capabilities in Web API 2, including: better error handling, attributed routes, CORS, and better OData support
* Understanding Web API controller activation
* Automatic lifetime management for database connections and transactions
* Using NHibernate, Entity Framework, and Dapper with Web API
* Easily secure a REST service, using standards-based authentication and authorization, JSON Web Tokens, or a custom implementation
* Supporting legacy SOAP callers with Web API
* How to expose relationships between resources
* Supporting partial resource updates under REST
* Web API versioning

### Detailed Table of Contents:

1. [ASP.NET](http://asp.net/) as a Service Framework

2. What is RESTful?

3. Designing Our Sample REST API

4. Building the Environment and Creating the Source Tree

5. Controllers, Dependencies, and Managing the Database Unit of Work

6. Securing the Service

7. Dealing with Relationships, Partial Updates, and Other Complexities

8. Leveraging new Web API Features to Simplify Your Code and Support Legacy Systems

9. Putting It All Together

### Time Frame for Completion:

Work on book throughout March and April and May

### Estimated Page Count:

Between 150 and 200

# Representative Feedback from Version 1

Comments from Amazon…

## Negative

* …degenerates into page after page of setting up NHibernate mappings, dependency injection, security and authentication, and, I kid you not, Log4Net logging. Page after page that could, and should have been used to show the ins and outs of the Web API, modeling a UI to utilize the RESTful links that are supposed to drive your application, crafting AJAX requests for PUT and DELETE, etc, etc, were instead used to fill up reams and reams of Log4Net XML setup, and DI mappings.
* You will spend most of your time learning about the authors favorite O/RM tools, Dependency Injection libraries and logging tools and how to configure them. You WILL NOT learn how to make REST calls using Ajax. I was looking to learning how to perform basic CRUD using a REST service without all the extraneous tools the authors covers. If you are looking to actually learn to use the Web API. DO NOT buy this book I repeat DO NOT by this book.
* Unfortunately, after stepping through a lot of complex project creation and configuration steps in Chapter 4, in Chapter 5 the author clearly gives up on the building, stops including steps with full sources, and instead refers you to download and explore his pre-created project.
* I was hoping to see a little more of a full fledge start to finish style book as the title as well as up to chapter 4 of the book suggest but that wasn't the case until you download the source code accompanying the book.
* Other than that though this book desperately needed someone to keep the author on track, and focused on what the book was supposed to be about: using the Web API.

Summary: Not focused on Web API, too many concepts, not enough step-by-step.

## Positive

* The book starts with a great and needed introduction of REST.
* This book is definitely absolutely utterly recommended for those who want to learn WebAPI MVC4 and also how to build a greatly architected WebAPI solution.

Summary: Keep the intro. Keep the good architectural practices.

## Suggestions

* I would also like to see how and where validation is done, and a chapter on how you would set up the same code for unit testing. I too would love to see an example of a simple html page consuming the newly built web service through a JavaScript Ajax call.
* And maybe a little more illustration of the authors examples might help out instead of relying on the book downloads
* This book would have been better if if was only 40 pages and just a step by step how to.
* I would be interested to see what the next book from this author would be like and hopefully it expands into some of the topics and concepts a little more in depth and sticks to the style that the first half of this book adheres to

Summary: Stick to highlighting the framework, show a client AJAX call, show [unit] testing, show step-by-step w/o *requiring* user to look at project code in github.

# Recommendation for Version 2

* Focus heavily on the framework. How to build on it. How to test what you’ve built.
* Show how to consume using an AJAX call(s). Don’t just rely on Fiddler. As we discovered with ccm, Fiddler hides issues you’ll encounter with real clients (e.g., CORS).
* Do NOT spend time on swappable data storage. Stick with one, probably NHibernate (though SQL Azure was requested by one guy).
* Do NOT spend time on OData. Nobody asked for it.
* Once we begin getting into discussing the implementation, let’s take vertical slices instead of horizontal slices (at least as much as possible). So, for example, show step-by-step how to GET a Task. Including testing and AJAX call.
* Similarly, let’s show the following in great detail, and only the following, rather than taking on explaining the whole task service. This focus will make it so readers don’t have to refer to the external project code to begin developing their own APIs. They can refer to that code to see all of the details if they want to, but most of the code is pretty much repetitive; after you’ve seen one GET you’ve seen enough to write another yourself.
  + GET Task(s)
  + POST Task(s)
  + PUT/PATCH Task(s)
  + DELETE Task(s)
  + GET Task User, to illustrate relationships
  + Perform a “non-resource” action (e.g., close a task).
  + Perform a legacy operation via the Web API.
    - Currently getting all Statuses and getting Status by id. May instead do this with Task, time permitting, b/c tasks are a bit more interesting.
* Drop Category and Priority. They may be good in a robust task tracking system, but they are overkill in illustrating Web API and good architectural practices. So get rid of any reference to them... globally.
* Table of Contents:
  + 1) ASP.NET as a Service Framework
    - Cut the ASP.NET MVC stuff. It’s confusing. For example, “This book will cover REST principles sufficiently for you to build services using ASP.NET MVC.” No, we’re not even going to use MVC. MVC and Web API are two different frameworks.
  + 2) What is RESTful?
    - Leave pretty much as-is, with the following changes. Readers rated this consistently well, even the ones who gave the book < 3 stars.
    - In CcmApi we're using 402 to indicate that a business processing rule has been violated. I think Adam Haskell or Jarrett recommended this, so I listened!
  + 3) Designing Our Sample REST API
    - Let's deal with handling non-resource APIs using REST. Specifically, starting, completing, or reopening a task. This time these will be different than simply updating or deleting it. There are business rules to follow. So the Status attribute is not editable via CRUD methods.
    - For collections, make the returned collection a "1st class object" with its own set of properties. Instead of putting the "All" link on each of the bare elements, put it on the collection which will now contain the elements. For the collections returning all Tasks or all Users (i.e., entities that can have a lot of records), add PageNumber, PageSize, and PageCount to the collection. Put pageNumber and pageSize in the query string so callers can page through results.
    - The new book and sample project code will use link services. This will help keep the list of links more dynamic than in the first book, where the mappers simply add a list of static links w/o any regard to current state.
    - ASP.NET Identity will be used instead of Membership Provider.
    - Good until the “Choosing Architecture Components” section. Stop there. Those details will move naturally into the text as needed, as we begin implementing vertical slices.
  + 4) Building the Environment and Creating the Source Tree
    - TODO
  + 5) Controllers, Dependencies, and Managing the Database Unit of Work
    - TODO
  + 6) Securing the Service
    - We should have 3 users with 3 different auth levels:
      * Do and see everything
      * Can't create or delete tasks, but can see and modify task data
      * Can't modify anything, and can't see who is assigned to the Task (e.g., for an external customer... don't want them to see which particular employee is working on their Task)
    - forms auth cookies not natural for apps and devices external to website, so will not cover it
  + 7) Dealing with Relationships, Partial Updates, and Other Complexities
    - TODO
  + 8) Leveraging new Web API Features to Simplify Your Code and Support Legacy Systems
  + 9) Putting It All Together
    - TODO

# Legacy SOAP Support

On a recent project we were tasked with creating a REST-based Web API for an existing banking system. There were a couple of notable requirements:

* The new REST-based API and the legacy SOAP-based web service must be packaged together in a single application.
* Existing customers must be able to benefit from the new features of the new application without affecting any external integration points. So, for example, a bank should be able to upgrade without introducing any breaking changes to its message translation service that bridges the Visa financial network and the banking system.

Understanding that SOAP requests are merely HTTP POST messages with an XML body, we were able to easily meet these requirements. An example illustrating how we did it will be the focus of this chapter.

## The Controller

Not needed! Just use a route handler. As an additional benefit, this skips the overhead of model binding and controller, action selection.

## Security Using a Route-Specific Handler

Route-specific handler

* <http://www.strathweb.com/2013/08/message-handlers-scoped-per-route-in-asp-net-web-api/>
* Using this for custom legacy auth. Does not burden other routes with overhead. Returns 401 (Unauthorized) if auth fails.

# Raw notes…

General

* Understand there are separate routes, controllers, etc. for mvc and web.api. Really confusing at first. Know which one you want to use.

Upgrading

<http://www.asp.net/mvc/tutorials/mvc-5/how-to-upgrade-an-aspnet-mvc-4-and-web-api-project-to-aspnet-mvc-5-and-web-api-2>

Versioning

* Use url-based versioning
* controller selector (namespaces)
  + Careful; this will break the api help page for example
  + Here’s a description of an impl, from msft: <http://blogs.msdn.com/b/webdev/archive/2013/03/08/using-namespaces-to-version-web-apis.aspx>
* Attribute based routing with constraint
  + More flexible
  + Unit testing???

Async Filters

* Be careful with this. Possible ctroller method invoked before filter has run. Not appropriate for uow or authorization filters!

CORS

Also, note that according to the CORS spec, setting origins to "\*" is not valid if **SupportsCredentials** is true (<http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api)>

IHttpActionResult

Security using web tokens

Auth attributes

Exposing entities and the risk of overposting. Recommend exposing (web) models instead of entities. This also provides greater flexibility (e.g., adding links).

Package management (nuGet) - when to add ref directly, when to use nuGet?

Partial updates thru patching

Idea from <http://weblog.west-wind.com/posts/2012/Aug/30/Using-JSONNET-for-dynamic-JSON-parsing>

Error handling / logging

* Use an ExceptionHandler
* Use an ExceptionLogger
* Use an ITraceWriter

Can get some ideas from here: <http://aspnetwebstack.codeplex.com/wikipage?title=Global%20Error%20Handling&referringTitle=Specs>

Formatters

* Applying security masks using custom formatter
* Look at global.asax ConfigureFormatters. We use JsonMediaTypeFormatter, CamelCasePropertyNamesContractResolver, StringEnumConverter. Can also enforce json-only by replacing the IContentNegotiator (e.g., JsonOnlyContentNegotiator).

Caching

<http://byterot.blogspot.com/2012/06/aspnet-web-api-caching-handler.html>