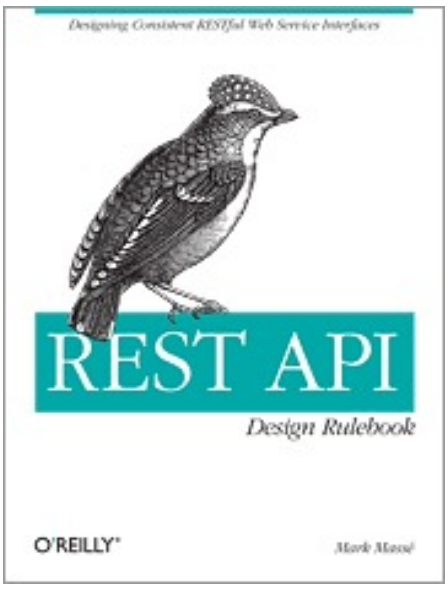

REST API Design Rulebook

Mark Masse

REST API Design Rulebook

Mark Masse

REST API Design Rulebook Mark Masse

In today's market, where rival web services compete for attention, a well-designed REST API is a must-have feature. This concise book presents a set of API design rules, drawn primarily from best practices that stick close to the Web's REST architectural style. Along with rules for URI design and HTTP use, you'll learn guidelines for media types and representational forms.

REST APIs are ubiquitous, but few of them follow a consistent design methodology. Using these simple rules, you will design web service APIs that adhere to recognized web standards. To assist you, author Mark Massé introduces the Web Resource Modeling Language (WRML), a conceptual framework he created for the design and implementation of REST APIs.

Learn design rules for addressing resources with URIs Apply design principles to HTTP's request methods and response status codes Work with guidelines for conveying metadata through HTTP headers and media types Get design tips to address the needs of client programs, including the special needs of browser-based JavaScript clients Understand why REST APIs should be designed and configured, not coded

REST API Design Rulebook Details

Date	:	Published (first published January 1st 2011)
ISBN	:	
Author	:	Mark Masse
Format	:	ebook 116 pages
Genre	:	Computer Science, Programming, Science, Technology, Nonfiction, Technical, Internet, Web

From Reader Review REST API Design Rulebook for online ebook

Kaido says

First part of the book gave me superb ideas about my API design but the second part was a bit too specific. Of course I agree that these things should be considered but as the frameworks out there, they are not very easy to bend and you have to follow the documentation as well in programming. otherwise good book.

Nicolás. says

Es un libro básico que see puede leer fácilmente en medio día o en un par de horas. Los primeros capítulos son esenciales, el libro empieza con la introducción de los "resources archetypes" (document, collection, store, and controller) en REST, un modelo que facilita el diseño de una API REST, simplemente recordando reglas básicas para cada uno de los arquetipos o unidades mínimas. Es un buen libro para principiantes o para quienes deseen recordar algunos aspectos básicos en la construcción de API'S REST. Del libro simplemente sacaría una lista de reglas que servirían como referencia. Acá comparto algunas de esas reglas:

REST API is composed of four distinct resource archetypes: document, collection, store, and controller. (el concepto de "store" creo es algo simplemente acuñado por el autor).

Some rules. Acá algunas reglas:

Rule: The query component of a URI should be used to paginate collection or store results.
Rule: A singular noun should be used for document names.
Rule: A plural noun should be used for collection names.
Rule: A plural noun should be used for store names.
Rule: A verb or verb phrase should be used for controller names.
Rule: Variable path segments may be substituted with identity-based values.
Rule: CRUD function names should not be used in URIs.
Rule: The query component of a URI may be used to filter collections or stores.
Rule: GET and POST must not be used to tunnel other request methods.
Rule: GET must be used to retrieve a representation of a resource.
Rule: HEAD should be used to retrieve response headers.
Rule: PUT must be used to both insert and update a stored resource.
Rule: PUT must be used to update mutable resources.
Rule: POST must be used to create a new resource in a collection.
Rule: POST must be used to execute controllers.
Rule: DELETE must be used to remove a resource from its parent.
Rule: OPTIONS should be used to retrieve metadata that describes a resource’s available interactions.

HTTP Codes. Reglas de respuesta.

Rule: 200 (“OK”) should be used to indicate nonspecific success.
Rule: 200 (“OK”) must not be used to communicate errors in the response body.
Rule: 201 (“Created”) must be used to indicate successful resource creation.
Rule: 202 (“Accepted”) must be used to indicate successful start of an asynchronous action.

Rule: 204 (“No Content”) should be used when the response body is intentionally empty.
Rule: 301 (“Moved Permanently”) should be used to relocate resources.
Rule: 302 (“Found”) should not be used.
Rule: 304 (“Not Modified”) should be used to preserve bandwidth.
Rule: 303 (“See Other”) should be used to refer the client to a different URI.
Rule: 400 (“Bad Request”) may be used to indicate nonspecific failure.
Rule: 401 (“Unauthorized”) must be used when there is a problem with the client’s credentials.
Rule: 403 (“Forbidden”) should be used to forbid access regardless of authorization state.
Rule: 404 (“Not Found”) must be used when a client’s URI cannot be mapped to a resource.
Rule: 405 (“Method Not Allowed”) must be used when the HTTP method is not supported.
Rule: 406 (“Not Acceptable”) must be used when the requested media type cannot be served.
Rule: 409 (“Conflict”) should be used to indicate a violation of resource state.
Rule: 500 (“Internal Server Error”) should be used to indicate API malfunction.

Others. Headers. Reglas.
Rule: Content-Type must be used.
Rule: Content-Length should be used.
Rule: Last-Modified should be used in responses.
Rule: ETag should be used in responses.
Rule: Stores must support conditional PUT requests. (the If-Unmodified-Sinceand/ or If-Matchrequest headers to express their intent...).
Rule: Stores must support conditional PUT requests A store resource uses the PUTmethod for both insert and update, which...
Rule: Location must be used to specify the URI of a newly created resource.
Rule: Cache-Control, Expires, and Date response headers should be used to encourage caching.
Rule: JSON should be supported for resource representation.
Rule: JSON must be well-formed.
Rule: A consistent form should be used to represent links.
Rule: CORS should be supported to provide multi-origin read/write access from JavaScript.

Y eso es todo, como vemos es un libro básico (y ademas publicado en el 2011), pero no por eso despreciable teniendo en cuenta que se trata de un libro fácil lectura. Enjoy.

Sergei Kasoverskij says

Good and succinct book to get a high level understanding of an good REST API.

Vladimir Miguro says

Somehow useful... There is a nice description about the types, controllers, methods, but I want more...
The authors concept of WMRL is too complex.

???? ??????? says

Good ideas to design rest api, readed in two evenings. Author use his framework for api desugn so that part of book where he explains his framework not neccesary to read.

Neville Ridley-smith says

Good as a reference. Weird if you read the whole thing.

A large chunk of the book - most of the second half - is about this thing called WRML. It's pretty useless as far as I can tell. It reminds me of other attempts to make things 'discoverable', like Jini or Soap directories. The fact of the matter is, you generally *never* want to dynamically interrogate a web api. You just want to use one to get something done. WRML may slightly help with this but not really - documentation and examples are always going to be better.

Anyway, the first half of the book is still really good in helping clarify best practices for REST API designs. Dip in and out as needed.

Robert says

Two huge problems with this book. It's short and very repetitive so the information content is about a couple of blog posts. The 'rules' are highly subjective, and much of the book is pushing the author's WRML 'standard' which I've never seen in the wild. The only real positive is that it's a comprehensive survey of the issues you need to think about when designing a REST API: just don't take the rules as gospel and research best practice from major APIs so you understand the context.

Rob says

If you're new to the world of REST APIs, and if you're looking for a good set of working rules on how to design them, then Mark Massé's *REST API Design Rulebook* should live up to its title just fine. It's a short book (you could read it in an afternoon) and it tackles the subject matter in a direct and orderly fashion. It starts with a brief history lesson, quickly reviewing the history of the world wide web, of HTTP, and of the emergence of the whole notion of "RESTful" APIs and services. From there, he lays out the six constraints of the web's architectural style, [1] and how RESTful designs fit into that. The rest of the book is basically just a series of "rules" to follow when designing a REST API, along with use-cases, examples, and justifications for each of those rules. Many of these rules seem very common sense and sound a lot like every other bit of advice you've ever received about (for example) naming variables and methods; [2] other rules seem sensible but get boxed in by "real-world" difficulties [3] or else seem counter to the prevailing wisdom; [4] still others seem to be little more than evangelizing for Massé's own proposed WRML design/modeling framework.

For me, the highlight reel included: the first three chapters (on identifier design, interaction design, and metadata design), and some of the discussion in the final chapter ("Client Concerns") about security, and the

overview (however brief) of JSONP and CORS as solutions for some otherwise challenging situations.

The two bits where it went off the rails a bit for me: (1) Any of the discussion of "hypermedia" and/or between-document linking--I haven't encountered this much (at all?) in the real-world and a lot of it seemed overkill to me, to go into that depth on how to link documents from within the API. [5] (2) WRML: what? Even after reading the whole book, I was still left wondering what exactly WRML was intended for, and what it would buy me as a developer. It seems that Massé wrote this book in part to evangelize this framework, but I could not quite get my mind sufficiently around it to say that I got it.

On the other hand, Massé does a great job with the "rules" (again: especially in the first half of the book) and I would recommend this to anyone that needs an introduction to REST API design. That being said, after absorbing this one, you'll probably want to explore further with one of the other O'Reilly books on the subject, and/or find yourself a couple of resources on OAuth.

(**Disclosure:** I received an electronic copy of this book from the publisher in exchange for writing this review.)

---

[1] (1) Client-server; (2) uniform interface; (3) layered system; (4) cache; (5) stateless; and (6) code-on-demand.

[2] *E.g.*, use singular nouns for documents; use plural nouns for collections and stores; use verbs ("and verb phrases") for actions, etc.

[3] In Chapter 2 ("Interaction Design with HTTP") Massé advises you to use the pedantically correct HTTP methods for performing the corresponding actions: *e.g.*, if you want to delete a document, send a DELETE request to the document's URI. *However*, as most web developers already know, mainstream browsers will not respect any HTTP methods other than GET or POST--so you can forget trying to use PUT or DELETE as a form's method, and you're stuck using POST as a wrapper. (Here is a good Stack Overflow thread on the subject.)

[4] *E.g.*, all those API URIs that have a '/v1/' in there somewhere? to specify the API version? Massé says: "Don't do that."

[5] And as an aside, it was very unclear to me how the rel (relationship) attributes were even suposed to be defined. Some central standard? Part of WRML? Do you make stuff up and furnish an end-point in your RESTful API to explain the rels?

Reese says

This book probably should have been titled 'Hypermedia REST API Design Rulebook', to prevent complaints from people who don't understand the real meaning of REST, but I don't know if that terminology even existed when this was published. At the very least, it should have more clearly separated the book into sections, stating the intended audiences explicitly. There is plenty that can be helpful to people writing a 'REST' API, and then there is a bunch of stuff that is useful to people trying to design a true hypermedia REST API, and finally, there is a fair amount that could be useful to someone writing a generic media type

specification (but most normal people wouldn't care to read). Most of the complaints I've seen are based on the subjects meant for the 2nd and 3rd audiences. Complaints about the inclusion of WRML media type details may be well founded, but the ideas behind this media type should definitely not be dismissed. As far as examples of good APIs, Twilio's API uses a media type that follows some of the principles behind WRML, and Github's uses a bunch as well.

Before this book, I was struggling to comprehend how to fulfill all the requirements of a true REST API. Including schema as a parameter in the media type filled the gap for me. I was also able to include many of the insights about using HTTP properly, like conditional PUT requests using headers, avoiding concurrency problems.

Christoffer Klang says

I skipped the WRML parts, but otherwise lots of good points.

Greenicicle says

The book tries to establish a set of rules for REST interfaces - I'm OK with that, it's what its title says. When you define rules, you take an opinion, that's fine. And most of what is defined as a rule is something I'd support as a good practice. A possible benefit of this rule book would be to provide some compact, memorizable, and clear guidance on good practices.

But then there's this obsession about WRML.

WRML is a proposed way of defining the content types and schemas of REST APIs - see http://www.wrml.org/about. And what do we find as the sole literature on WRML: the REST API Design Rulebook. WRML has about zero industry acceptance; I never heard of it, and it does not even have a wikipedia page. And yet the REST API Design Rulebook puts it into the heart and center of every halfway advanced topic. Now a "rule" that is centered around an obscure standard proposal is totally void, and so this book is a simple waste of e-ink (even if that was its only fault, which it isn't).

Thomas says

Nice thoughts, but feels like an academic paper out of touch with modern dev.

Ahmad Hajjar says

Until the day before reading this book, I had this misunderstanding about RESTful API/Services, a lot of services out there, claiming to be RESTful are just claiming so because they use HTTP as the communication media, while this is only the beginning, to be RESTful the application has to comply with HATEOAS, using Hypermedia As The Engine of Application State is the thing that makes a RESTful app, a RESTful app.

However, the author is exaggerating in using and/or emphasising on the use of links and relations in the book's examples, I see this as a good thing to showcase how a RESTful representation should be, but IMHO I see this as overkill. But if we take it from another angle, standardising could always be the answer to a lot of this industry's problems. Here in this case, standardisation can help the client's program (if they follow the REST Standard) to use the application and navigate through it, and any of its versions, by just knowing the rootdoc. In this case, the client's app will use the API just like any human who visiting a Website and "navigating" using hyperlinks, and any changes in the server side (resources location, objects representations, or behaviours) will not affect the clients, in this perfect world the server developers and client developers will be totally independent :) they don't even share "API documentation" because in this case the API is describing itself.

Tihomir says

Good, but very short book.

Michael says

This is a better book than my pure star rating would seem to indicate. It's not that I didn't like the book - it was OK.

However, as a highly opinionated book, I wonder how widely accepted the ideas being laid down as rules actually are. I think there is much here that people can agree is best practice when designing a RESTful API. However, I'm left with the feeling that much of the rest of it is a one-man campaign towards standards that may never be adopted at large.

So, reader beware: I suspect these "rules" are pretty solid for the author, but I don't know how widely accepted they are in the industry at large quite yet.
