How to design RESTful search/filtering?

I'm currently designing and implementing a RESTful API in PHP. However, I have been unsuccessful implementing my initial design.

```
GET /users # list of users
GET /user/1 # get user with id 1
POST /user # create new user
PUT /user/1 # modify user with id 1
DELETE /user/1 # delete user with id 1
```

So far pretty standard, right?

My problem is with the first one `GET /users`. I was considering sending parameters in the request body to filter the list. This is because I want to be able to specify complex filters without getting a super long url, like:

```
GET /users?parameter1=value1&parameter2=value2&parameter3=value3&parameter4=value4
```

Instead I wanted to have something like:

```
GET /users
# Request body:
{
    "parameter1": "value1",
    "parameter2": "value2",
    "parameter3": "value3",
    "parameter4": "value4"
}
```

which is much more readable and gives you great possibilities to set complex filters.

Anyway, `file_get_contents('php://input')` didn't return the request body for `GET` requests. I also tried `http_get_request_body()`, but the shared hosting that I'm using doesn't have `pecl_http`. Not sure it would have helped anyway.

I found this question and realized that GET probably isn't supposed to have a request body. It was a bit inconclusive, but they advised against it.

So now I'm not sure what to do. How do you design a RESTful search/filterng function?

I suppose I could use `POST`, but that doesn't seem very RESTful.

design api search rest filter

edited May 23 at 12:34
Community ♦
1 • 1

asked Feb 16 '11 at 18:45
Erik B
19.8k • 19 • 74 • 98

6 possible duplicate of RESTful URL design for search – outis Mar 2 '12 at 11:03

27 Be careful!!! GET method must be IDEMPOTENT, and must be "cacheable". If you send information in the body How can the system cache your request? HTTP allows caching GET request using only the URL, not the request body. For instance, this two requests: example.com { test:"some" } example.com { anotherTest:"some2" } are considered the same by the cache system: Both of them have exactly the same URL – jfcorugedo Feb 23 '15 at 11:45

7 Just to add, you should POST to the /users (collection) and not /user (single user). – Mladen B. Jun 24 '15 at 11:34

7 Answers

The best way to implement a RESTful search is to consider the search itself to be a resource. Then you can use the POST verb because you are creating a search. You do not have to literally create something in a database in order to use a POST.

For example:

```
Accept: application/json
Content-Type: application/json
POST http://example.com/people/searches
```

```
{
  "terms": {
    "ssn": "123456789"
  },
  "order": { ... },
  ...
}
```

You are creating a search from the user's standpoint. The implementation details of this are irrelevant. Some RESTful APIs may not even need persistence. That is an implementation detail.

edited Mar 20 '14 at 5:11
jdphenix
8,951 ● 2 ● 27 ● 55

answered Sep 21 '13 at 14:39
Jason Harrelson
3,301 ● 1 ● 6 ● 2

132 One significant limitation to using a POST request for a search endpoint is that it cannot be bookmarked. Bookmarking search results (particularly complex queries) can be quite useful. – couchand Jan 17 '14 at 20:57

43 Using POST to make searches may break the REST cache constraint. whatisrest.com/rest_constraints/cache_excerps – Filipe Jan 18 '14 at 12:21

28 Searches, by their nature, are transient: data evolves between two searches with the same parameters, so I think that a GET request does not map cleanly to the search pattern. Instead, the search request should be POST (/Resource/search), then you can save that search and redirect to a search result, e.g. /Resource/search/iyn3zrt. That way, GET requests succeed and make sense. – sleblanc Apr 28 '14 at 17:53

19 I don't think post is suitable method for searching, data for normal GET requests could vary over time too. – wonder May 29 '15 at 9:34

26 This is absolutely the worst possible answer. I can't believe it has so many upvotes. This answer explains why: programmers.stackexchange.com/questions/233164/… – richard Feb 13 '16 at 6:06

If you use the request body in a GET request, you're breaking the REST principle, because your GET request won't be able to be cached, because cache system uses only the URL.

And what's worse, your URL can't be bookmarked, because the URL doesn't contains all the information needed to redirect the user to this page

Use URL or Query parameters instead of request body parameters.

e.g.:

```
/myapp?var1=xxxx&var2=xxxx
/myapp;var1=xxxx/resource;var2=xxxx
```

In fact, the HTTP RFC 7231 says that:

A payload within a GET request message has no defined semantics; sending a payload body on a GET request might cause some existing implementations to reject the request.

For more information take a look at here

edited Sep 16 '16 at 13:56

answered Feb 23 '15 at 11:53
jfcorugedo
4,096 ● 2 ● 23 ● 33

3 Learn from my mistake - I designed an api using the accepted answer's suggestion (POSTing json), but am moving over to url parameters. Bookmark-ability may be more important than you think. In my case, there was a need to direct traffic to certain search queries (ad campaign). Also, using the history API makes more sense with URL parameters. – Jake Aug 16 at 13:57

1 THIS is this the correct answer. – pimbrouwers Sep 6 at 14:12

It seems that resource filtering/searching can be implemented in a RESTful way. The idea is to introduce a new endpoint called `/filters/` or `/api/filters/`.

Using this endpoint *filter* can be considered as a resource and hence created via `POST` method. This way - of course - body can be used to carry all the parameters as well as complex search/filter structures can be created.

After creating such filter there are two possibilities to get the search/filter result.

1. A new resource with unique ID will be returned along with `201 Created` status code. Then using this ID a `GET` request can be made to `/api/users/` like:

   ```
   GET /api/users/?filterId=1234-abcd
   ```

2. After new filter is created via `POST` it won't reply with `201 Created` but at once with `303 SeeOther` along with `Location` header pointing to `/api/users/?filterId=1234-abcd`. This redirect will be automatically handled via underlying library.

In both scenarios two requests need to be made to get the filtered results - this may be considered as a drawback, especially for mobile applications. For mobile applications I'd use

single `POST` call to `/api/users/filter/` .

How to keep created filters?

They can be stored in DB and used later on. They can also be stored in some temporary storage e.g. redis and have some TTL after which they will expire and will be removed.

What are the advantages of this idea?

Filters, filtered results are cacheable and can be even bookmarked.

edited Nov 14 at 14:41

answered Nov 26 '15 at 20:46

Opal
42.3k ● 20 ● 74 ● 91

I think you should go with request parameters but only as long as there isn't an appropriate HTTP header to accomplish what you want to do. The HTTP specification does not explicitly say, that GET can not have a body. However this paper states:

> By convention, when GET method is used, all information required to identify the resource is encoded in the URI. There is no convention in HTTP/1.1 for a safe interaction (e.g., retrieval) where the client supplies data to the server in an HTTP entity body rather than in the query part of a URI. This means that for safe operations, URIs may be long.

answered Feb 16 '11 at 19:02

Daff
32.7k ● 6 ● 74 ● 99

ElasticSearch also does GET with body and works well! – Tarun Sapra Jan 4 '16 at 16:34

Yeah but they control the server implementation may not be tne xase on the interwebs. – user432024 Apr 28 at 3:35

Don't fret too much if your initial API is fully RESTful or not (specially when you are just in the alpha stages). Get the back-end plumbing to work first. You can always do some sort of URL transformation/re-writing to map things out, refining iteratively until you get something stable enough for widespread testing ("beta").

You can define URIs whose parameters are encoded by position and convention on the URIs themselves, prefixed by a path you know you'll always map to something. I don't know PHP, but I would assume that such a facility exists (as it exists in other languages with web frameworks):

.ie. Do a "user" type of search with param[i]=value[i] for i=1..4 on store #1 (with value1,value2,value3,... as a shorthand for URI query parameters):

```
1) GET /store1/search/user/value1,value2,value3,value4
```

or

```
2) GET /store1/search/user,value1,value2,value3,value4
```

or as follows (though I would not recommend it, more on that later)

```
3) GET /search/store1,user,value1,value2,value3,value4
```

With option 1, you map all URIs prefixed with `/store1/search/user` to the search handler (or whichever the PHP designation) defaulting to do searches for resources under store1 (equivalent to `/search?location=store1&type=user` .

By convention documented and enforced by the API, parameters values 1 through 4 are separated by commas and presented in that order.

Option 2 adds the search type (in this case `user`) as positional parameter #1. Either option is just a cosmetic choice.

Option 3 is also possible, but I don't think I would like it. I think the ability of search within certain resources should be presented in the URI itself preceding the search itself (as if indicating clearly in the URI that the search is specific within the resource.)

The advantage of this over passing parameters on the URI is that the search is part of the URI (thus treating a search as a resource, a resource whose contents can - and will - change over time.) The disadvantage is that parameter order is mandatory.

Once you do something like this, you can use GET, and it would be a read-only resource (since you can't POST or PUT to it - it gets updated when it's GET'ed). It would also be a resource that only comes to exist when it is invoked.

One could also add more semantics to it by caching the results for a period of time or with a DELETE causing the cache to be deleted. This, however, might run counter to what people typically use DELETE for (and because people typically control caching with caching headers.)

How you go about it would be a design decision, but this would be the way I'd go about. It is not perfect, and I'm sure there will be cases where doing this is not the best thing to do (specially for very complex search criteria).

edited Jan 26 at 14:38

answered Feb 16 '11 at 19:13

luis.espinal
7,399 ● 4 ● 25 ● 48

6 Yo, if you (someone, whoever/whatever) things approprite to downvote my answer, would it hurt you ego to at least put a comment indicating what exactly do you disagree with? I know it's the interweebz, but ... ;) – luis.espinal Apr 10 '12 at 15:53

72 I didn't downvote, but the fact that the question starts with: "I'm currently designing and implementing a RESTful API" and your answer starts with "Don't fret too much if your initial API is fully RESTful or not" feels wrong to me. If you're designing an API you are designing an API. The question is asking how to best design the API, not about whether the API should be designed. – gardarh Oct 31 '12 at 15:23

11 The API *is* the system, work on the API first, not the backend plumbing, the first implementation could/should just be a mock. HTTP has a mechanism for passing parameters, you are suggesting it be reinvented, but worse (ordered parameters instead of name value pairs). Hence the down vote. – Steven Herod Jul 1 '13 at 5:31

10 @gardarh - yes, it feels wrong, but at times, it is pragmatic. The primary objective is to design an API that works for the business context at hand. If a fully RESTFULL approach is appropriate to the business at hand, then go for it. If it is not, then don't go for it. That is, design an API that meets your specific business requirements. Going around trying to make it RESTfull as its primary requirement is no much different from asking "how do I use the adapter pattern in X/Y problem." Don't shoe horn paradigms unless they solve actual, valuable problems. – luis.espinal Jul 1 '13 at 20:53

1 I view a resource as some collection of state, and parameters as a means for manipulating the representation of that state parametrically. Think of it this way, if you could use knobs and switches to adjust how the resource is displayed (show/hide certain bits of it, order it differently, etc...) those controls are params. If it's actually a different resource ('/albums' vs '/artists', for instance), that's when it should be represented in the path. That's what is intuitive to me, anyway. – Eric Elliott Oct 27 '13 at 6:15

FYI: I know this is a bit late but for anyone who is interested. Depends on how RESTful you want to be, you will have to implement your own filtering strategies as the HTTP spec is not very clear on this. I'd like to suggest url-encoding all the filter parameters e.g.

```
GET api/users?filter=param1%3Dvalue1%26param2%3Dvalue2
```

I know it's ugly but I think it's the most RESTful way to do it and should be easy to parse on the server side :)

answered Apr 21 at 11:02

shanks
489 ● 5 ● 22

As I'm using a laravel/php backend I tend to go with something like this:

/resource?filters[status_id]=1&filters[city]=Sydney&page=2&include=relatedResource

PHP automatically turns [] params into an array, so in this example I'll end up with a $filter variable that holds an array/object of filters, along with a page and any related resources I want eager loaded.

If you use another language, this might still be a good convention and you can create a parser to convert [] to an array.

answered Jun 20 at 8:53

the-a-train
454 ● 5 ● 23

1 Why the downvote? – the-a-train Nov 7 at 2:03