REST Resource Naming Guide

In REST, primary data representation is called **Resource**. Having a strong and consistent REST resource naming strategy – will prove one of the best design decisions in the long term.

> *The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g., a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.*
>
> — Roy Fielding's dissertation

A **resource can be a singleton or a collection**. For example, "`customers`" is a collection resource and "`customer`" is a singleton resource (in a banking domain). We can identify "`customers`" collection resource using the URI "`/customers`". We can identify a single "`customer`" resource using the URI "`/customers/{customerId}`".

A **resource may contain sub-collection resources** also. For example, sub-collection resource "`accounts`" of a particular "`customer`" can be identified using the URN "`/customers/{customerId}/accounts`" (in a banking domain). Similarly, a singleton resource "`account`" inside the sub-collection resource "`accounts`" can be identified as follows: "`/customers/{customerId}/accounts/{accountId}`".

REST APIs use Uniform Resource Identifiers (URIs) to address resources. REST API designers should create URIs that convey a REST API's resource model to its potential client developers. When resources are named well, an API is intuitive and easy to use. If done poorly, that same API can feel difficult to use and understand.

The constraint of a uniform interface is partially addressed by the combination of URIs and HTTP verbs and using them in line with the standards and conventions.

Below are a few tips to get you going when creating the resource URIs for your new API.

REST Resource Naming Best Practices

Use nouns to represent resources

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties which verbs do not have – similar to resources have attributes. Some examples of a resource are:

- Users of the system
- User Accounts

- Network Devices etc.

and their resource URIs can be designed as below:

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices/{device-id}
http://api.example.com/user-management/users/
http://api.example.com/user-management/users/{id}
```

For more clarity, let’s divide the **resource archetypes** into four categories (document, collection, store and controller) and then **you should always target to put a resource into one archetype and then use it’s naming convention consistently**. *For uniformity’s sake, resist the temptation to design resources that are hybrids of more than one archetype.*

1. **document**

   A document resource is a singular concept that is akin to an object instance or database record. In REST, you can view it as a single resource inside resource collection. A document’s state representation typically includes both fields with values and links to other related resources.

   Use “singular” name to denote document resource archetype.

   ```
   http://api.example.com/device-management/managed-devices/{device-id}
   http://api.example.com/user-management/users/{id}
   http://api.example.com/user-management/users/admin
   ```

2. **collection**

   A collection resource is a server-managed directory of resources. Clients may propose new resources to be added to a collection. However, it is up to the collection to choose to create a new resource or not. A collection resource chooses what it wants to contain and also decides the URIs of each contained resource.

   Use the “plural” name to denote the collection resource archetype.

   ```
   http://api.example.com/device-management/managed-devices
   http://api.example.com/user-management/users
   http://api.example.com/user-management/users/{id}/accounts
   ```

3. **store**

   A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them. A store never generates new URIs. Instead, each stored resource has a URI. The URI was chosen by a client when it was initially put into the store.

   Use “plural” name to denote store resource archetype.

   ```
   http://api.example.com/song-management/users/{id}/playlists
   ```

4. **controller**

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs.

Use “verb” to denote controller archetype.

```
http://api.example.com/cart-management/users/{id}/cart/checkout
http://api.example.com/song-management/users/{id}/playlist/play
```

Consistency is the key

Use consistent resource naming conventions and URI formatting for minimum ambiguily and maximum readability and maintainability. You may implement below design hints to achieve consistency:

1. **Use forward slash (/) to indicate hierarchical relationships**

   The forward slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources. e.g.

   ```
   http://api.example.com/device-management
   http://api.example.com/device-management/managed-devices
   http://api.example.com/device-management/managed-devices/{id}
   http://api.example.com/device-management/managed-devices/{id}/scripts
   http://api.example.com/device-management/managed-devices/{id}/scripts/{id}
   ```

2. **Do not use trailing forward slash (/) in URIs**

As the last character within a URI's path, a forward slash (/) adds no semantic value and may cause confusion. It's better to drop them completely.

```
http://api.example.com/device-management/managed-devices/
http://api.example.com/device-management/managed-devices 	/*This is much better version*/
```

3. **Use hyphens (-) to improve the readability of URIs**

To make your URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments.

```
http://api.example.com/inventory-management/managed-entities/{id}/install-script-location  //More readable
http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation  //Less readable
```

4. **Do not use underscores (_)**

It's possible to use an underscore in place of a hyphen to be used as separator – But depending on the application's font, it's possible that the underscore (_) character can either get partially obscured or completely hidden in some browsers or screens.

To avoid this confusion, use hyphens (-) instead of underscores (_).

```
http://api.example.com/inventory-management/managed-entities/{id}/install-script-location
//More readable
http://api.example.com/inventory_management/managed_entities/{id}/install_script_location
//More error prone
```

5. **Use lowercase letters in URIs**

When convenient, lowercase letters should be consistently preferred in URI paths.

RFC 3986 defines URIs as case-sensitive except for the scheme and host components. e.g.

```
http://api.example.org/my-folder/my-doc  //1
HTTP://API.EXAMPLE.ORG/my-folder/my-doc  //2
http://api.example.org/My-Folder/my-doc  //3
```

In above examples, 1 and 2 are same but 3 is not as it uses **My-Folder** in capital letters.

6. **Do not use file extentions**

File extensions look bad and do not add any advantage. Removing them decreases the length of URIs as well. No reason to keep them.

Apart from above reason, if you want to highlight the media type of API using file extenstion then you should rely on the media type, as communicated through the `Content-Type` header, to determine how to process the body’s content.

```
http://api.example.com/device-management/managed-devices.xml  /*Do not use it*/
http://api.example.com/device-management/managed-devices      /*This is correct URI*/
```

Never use CRUD function names in URIs

URIs should not be used to indicate that a CRUD function is performed. URIs should be used to uniquely identify resources and not any action upon them. HTTP request methods should be used to indicate which CRUD function is performed.

```
HTTP GET http://api.example.com/device-management/managed-devices  //Get all devices
HTTP POST http://api.example.com/device-management/managed-devices  //Create new Device

HTTP GET http://api.example.com/device-management/managed-devices/{id}  //Get device for given Id
HTTP PUT http://api.example.com/device-management/managed-devices/{id}  //Update device for given Id
HTTP DELETE http://api.example.com/device-management/managed-devices/{id}  //Delete device for given Id
```

Use query component to filter URI collection

Many times, you will come across requirements where you will need a collection of resources sorted, filtered or limited based on some certain resource attribute. For this, do not create new APIs – rather enable sorting, filtering and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices?region=USA
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
http://api.example.com/device-management/managed-devices?
region=USA&brand=XYZ&sort=installation-date
```

Was this article helpful?

YES NO

Previous Tutorial:
REST API Versioning

Next Tutorial:
REST API Security Essentials

ADVERTISEMENTS

Comments

Mario says
July 3, 2020 at 10:07 pm

Great article I have a doubt I have the next endpoints:

localhost:8080/api/product: //can return one or more products given an array of ids

maps to : (@RequestBody List<IdContainer> companyId)

localhost:8080/api/product: // return 0 or N products

maps to: getProductsAll() ;

How should I name each endpoint?

Reply

Admin says
July 9, 2020 at 7:06 am

`HTTP GET localhost:8080/api/products` shall return all products. For a set of product ids, pass them as query parameter to filter i.e. `HTTP GET localhost:8080/api/products?ids=1,2,3,4`

Reply

Zac says
May 19, 2020 at 2:51 pm

Hello,

I'm a part-time developer, so I don't keep up with these types of issues on a regular basis:

Is there any discussion of an API standards definition for base object classes? For example, using this site's terminology, will there ever be a standard Document for a Person base class, Location base class, etc?

I find it interesting that almost every API I've developed has a couple of Document types that are reused to the point that I just copy/paste from my personal library when they're needed. It would be nice if there was a universally recognized Person class that I could extend to meet my needs. More importantly, if I consumed someone else's API, I would know that their RegisteredUser class, for example, extends this Person base class as well.

In my opinion, APIs are THE next major advancement interoperability, personal devices/cool-bells-and-whistles notwithstanding. I think we're still in the Wild West days of API technology but I would love to see some structure put around this, if it doesn't already exist.

Regards

Zac

Reply

shaktimaan says
May 16, 2020 at 12:39 pm

Really helpful article, thank you for taking your time and writing this.

I have a question regarding naming of resource. If I want to fetch all the document-links in a document based on the type and version.

Option 1: GET /documents/{type}/links?version=x&other=params

Option 2: GET /documents/{type}/{version}/links?other=params

Which endpoint naming should I follow?

Reply

Andrii says
May 18, 2020 at 9:16 pm

Hello. I think the better option will be `GET http://documents/{document-id}/document-links?type=YOUR-TYPE&version=YOUR-VERSION`

Ps. You can simplify it to links from document-links, as you want.

Hope, it will be helpful for you 🙂

Reply

Niklas says
June 11, 2020 at 2:49 pm

Hi.

First. A very good guide on naming! Thanks.

It is worth mentioning that hierarchical URLs may lead to problems if we (possibly at a later stage) want to enable filtering that span document-links of different document-ids. How can we represent such collection?

Alternative 1 – use some wildcard notation (*):

`http://documents/*/document-links?type=YOUR-TYPE&version=YOUR-VERSION`

The ‘*’ would need to be escaped and it is hard to find a good word that is intuitive. Maybe ‘null’ or ‘any’ would work?

`http://documents/null/document-links?type=YOUR-TYPE&version=YOUR-VERSION`

`http://documents/any/document-links?type=YOUR-TYPE&version=YOUR-VERSION`

…but this is a bit unconventional and not intuitive. We will not get support in standards like OpenAPI and in many server-side implementation frameworks you would have to do some workaround because it does not fit with out-of-the-box functionality.

Alternative 2 – have all collections on the same level

`http://documents/{id}`

`http://document-links/{id}`

A search that spans multiple documents:

`http://document-links?type=YOUR-TYPE&version=YOUR-VERSION`

...and we would still be able to filter inside a specific document (documentId=abc123) by adding one more filter:

```
http://document-links?documentId=abc123&type=YOUR-TYPE&version=YOUR-VERSION
```

With this URL style we are able to introduce filtering based on fields without restricting them to be inside a specific document:

```
http://document-links?type=YOUR-TYPE&version=YOUR-VERSION
```

and are still able to filter within the scope of a specific document

```
http://document-links?documentId=1type=YOUR-TYPE&version=YOUR-VERSION
```

The hierarchical URL style is a bit easier to read but putting all collections on the same level makes it easier to extend the API with new filtering options and thus may be more suited for future requirements.

Reply

Michael says
May 15, 2020 at 7:16 pm

Thank you for a very interesting article. One of the very few that talk about a controller archetype.

Here is my question to you. Let's say you want to define the end-points for a College. You have a list of all courses available to the students for Fall 2019. This includes the course title, description, number of credits, pre-requisits … etc. The URI would would be https://api.mycollegesite.com/courses/2019/fall

At the same time, you want to access the courses for which a particular student has registered for fall 2019. Those would include the ID of the course, the grade, the grades for each homework, grades for each exam …

The URI would be https://api.mycollegesite.com/students/123456/courses/2019/fall

Given that the resource returned by the first end-point is substantially different from the resource returned by the second end-point, should the resource name "courses" be used in both or is it better to have different resoure names?

Would be better to replace the first URI by

https://api.mycollegesite.com/courses/curriculum/2019/fall or

https://api.mycollegesite.com/curriculum/courses/2019/fall

Reply

Sam says
June 2, 2020 at 8:15 pm

First URI: Courses seems to be the “collection” and year & semester are filters.

```
/courses?year=2019&semester=fall
```

Second URI: Grades belong to each student so they could be a collection under students. These are filtered by semester. Grades could also be their own collection (maybe you want to get all grades for visualization?). If it’s useful for grades to exist independent from students, you can make grades their own collection.

```
/students/{id}/grades?year=2019&semester=fall
```

```
/grades?student_id=123&year=2019&semester=fall
```

Reply

Seb says
May 12, 2020 at 7:12 am

Hi there, Thanks for this rules list.

I was wondering is there a naming convention for secured resources (endpoint protected by login/password ?)

Reply

Alexander says
May 15, 2020 at 9:32 am

I think, those aforementioned conventions are enough, so as to comply with uniform interface constraint. The difference is to access secured resources, it must follow authentication mechanism which is a common practice.

Reply

XinyuZhou says

May 7, 2020 at 3:34 am

Excuse me, I have a question about hierarchical relationship. How to define the number of layers, whether the parameter in the path are counted as one layer.

For example, if “/device-management/managed-devices” has two hierarchies, then how many hierarchies in “/device-management/managed-devices/{id}”? three? or still two?

Reply

Alexander says
May 15, 2020 at 9:51 am

I am curious to know why you want to count the number of layers in the URI. But, anyway, for the given case, there are 3 layers, as the forward slashes are used to define relationships.

“/device-management/managed-devices” is a collection

while

“/device-management/managed-devices/{id}” is a document (per definition of resource archetype)

Reply

llw says
April 16, 2020 at 2:18 pm

I think it is fair for the store archetype to use singular nouns. It is not designed to be an access point for individual items in it, which means URL pattern like the following is not used in general:

…/store/{item-id}

On the other hand, using "users/{id}/carts" leaves the impression that a user may have multiple carts. In fact, in your later controller examples, you used singular nouns (cart and playlist) for store as well:

http://api.example.com/cart-management/users/{id}/cart/checkout

http://api.example.com/song-management/users/{id}/playlist/play

Reply

Alexander says
May 15, 2020 at 9:59 am

Nice catch on the two examples of controller resource. I wait for explanation/clarification of the admin, in case he had typos in those examples.

Reply

Admin says
May 15, 2020 at 11:28 am

I think I picked up the incorrect example. Fixed it.

Reply

May says
March 31, 2020 at 3:03 pm

Hey,

Regarding using query component to filter URI collection – how should I do it if I want only managed-devices that has region field exists? Not specific region=USA, only that region exists?

Reply

Alexander says
May 15, 2020 at 10:14 am

Maybe you can use a specific parameter to denote that the parameter is to be used for region field existence, e.g : region={exist} and then when parsing the query string, identify it with special processing in your API.

Reply

Gökhan Ayrancıoğlu says
March 25, 2020 at 9:14 am

Hi,

Great article thank you,

I have a question for naming of resource:

– There are two api in my project. One of them is sending single message to my service. The other one sending multiple message to my service. How to naming of them correctly ?

I am using this one, but i am sure there is better way to naming them.

POST: api/v1/tickets/message // Single message

POST api/v1/tickets/messages // Multiple message

Reply

Admin says
March 25, 2020 at 6:15 pm

Without understanding the whole use-case, it would not be correct to suggest appropriate naming. Though, one suggestion is to have only one API `"POST api/v1/tickets/messages"`. It should be accept 1 to N messages.

Reply

Chris says
April 21, 2020 at 4:14 am

“tickets/messages” — collection in a collection, shouldn’t be like this “tickets/{id}/messages?action=sendmultiple

sendsingle”

Reply

Girish says
March 6, 2020 at 10:29 am

How to name Controller ? eg. If i have resource(Folder) called ‘Customers’ inside this should I create controller called CustomerController or CustomersConstroller ?

Reply

Admin says
March 6, 2020 at 3:32 pm

Focus on resource naming. Here, the primary resource is “Customer” so `CustomerController` is a better name. “Customers” is like a collection of “Customer” resources.

Reply

Taco Tijsma says
January 28, 2020 at 10:14 am

We use string type ID’s which could include special characters like the forward slash. A typical Url looks like this :

GET https://hostname/api/v1/resource/AB/124747 / B1

The spaces and forward slashes suppose to be there. The id in this example is: AB/124747 / B1

Is there a way to deal with this type of id’s? Or do we have to use querystring type Url like:

GET https://hostname/api/v1/resource?id=AB/124747 / B1

Reply

Admin says
January 28, 2020 at 5:09 pm

I will suggest to consider a design change and replace "/" with another character e.g. hyphen, underscore or hash.

Reply

Djulio says
January 30, 2020 at 9:47 am

If you need to keep the "/", just replace it with something else for your request, as Admin suggested, and then when you retrieve it, parse it and replace that other symbol with "/" again. Example: "AB/124747 / B1" => in your request: "AB@124747 @ B1" => after reading it from your request, transform it back to: "AB/124747 / B1"

Reply

alejo4373 says
February 26, 2020 at 5:41 pm

Couldn't you just URL encode your id? So your id `/AB/124747 / B1` will become `AB%2F124747%20%2F%20B1`

Reply

Josef Sábl says
January 23, 2020 at 2:12 pm

Should the store endpoint have the GET method that returns everything stored or is it up to the client to "remember" the ids?

Reply

Reiner Saddey says
May 19, 2019 at 5:18 pm

There may be another practical reason to avoid trailing slashes.

They might just not be supported by a particular framework, e.g. JAX-RS.

See stackoverflow – rest – RESTEasy cutting trailing slash off @Path at https://stackoverflow.com/questions/15196698/rest-resteasy-cutting-trailing-slash-off-path

Reply

Arthur says
May 5, 2019 at 8:58 pm

This page presents a common myth of CRUD & URIs.

Roy Fielding writes:

A REST API must not define fixed resource names or hierarchies (an obvious coupling of client and server). Servers must have the freedom to control their own namespace. Instead, allow servers to instruct clients on how to construct appropriate URIs, such as is done in HTML forms and URI templates, by defining those instructions within media types and link relations.

https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

Reply

Admin says
May 6, 2019 at 4:28 pm

Thanks for sharing your thoughts. It is right that URIs should be instructed to clients through hypermedia.

“Servers must have the freedom to control their own namespace” – but still there MUST be some design at server side which will create those hypermedia links in some standard structure. Or please share an example if there is a gap.

Reply

Arthur says
May 20, 2019 at 2:52 am

If you use hypermedia in API, syle of URI all the more doesn't matter for clients or users. Fielding confirms this in his dissertation:

At no time whatsoever do the server or client software need to know or understand the meaning of a URI — they merely act as a conduit through which the creator of a resource (a human naming authority) can associate representations with the semantics identified by the URI.

https://www.ics.uci.edu/~fielding/pubs/dissertation/evaluation.htm#sec_6_2_4

He also said: There is no such thing as a REST endpoint. There are resources. A countably infinite set of resources bound only by restrictions on URL length.

https://twitter.com/fielding/status/1052976631374000128

Next, URI does not limit the scope of what might be a resource; rather, the term "resource" is used in a general sense for whatever might be identified by a URI.

Finally, these APIs is RESTful? Or not? Please, provide argumentation.

https://developers.facebook.com/docs/graph-api/

https://api.slack.com/methods

https://www.flickr.com/services/api/

https://core.telegram.org/methods

https://developer.twitter.com/en/docs/api-reference-index.html

In general, all this talk of URI names is no more than REST buzzwording.

Reply

Andrew Boyd says
January 25, 2020 at 1:12 pm

A REST API must not define fixed resource names or hierarchies (an obvious coupling of client and server).

You me like this->https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

rest-apis-must-be-hypertext-driven seems like a fixed resource name to me.

Reply

Jaya says
April 3, 2019 at 11:23 am

Let us say there is a single resource that we need to retrieve via two unique references (id or code) separately:

*. Option 1.

/v2/suppliers/{id}

/v2/suppliers/{code}

*. Option 2.

/v2/suppliers/{id}

/v2/suppliers?code={code}

So from the above please let me know the preferred option or is there another way?

Thanks.

Reply

Toni Maunde says
April 17, 2019 at 4:58 pm

My question would be the reason why you have two URIs for the same resource? I'm here thinking about your database and how you have two unique identifiers for the same entity.

TLDR: drop the /{code} or drop the /{id}. Use one of them. Query strings are meant to sort collections, not to get a single document.

Reply

David Yazzie says
April 17, 2019 at 11:31 pm

Option 1 cant work as the API wouldnt know which field to parse as. You could do

/v2/suppliers/id/{id}

/v2/suppliers/code/{code}

Reply

Frans says
September 12, 2019 at 3:18 pm

No! What kind of resource is /v2/suppliers/id in this example? Or /v2/suppliers/code ? That’s right, there’s no good answer to that, because this is bad use of REST.

Toni Maunde’s was just fine. Pick your identifier and stick with it.

Reply

Wil says

September 24, 2019 at 7:41 pm

Let’s critical think it through,

/v2/suppliers/id/{id}

/v2/suppliers/code/{code}

Is there a resource collection: /v2/suppliers/id or /v2/suppliers/code? Probably not — that’s an oops. {id} and {code} have no collection to belong to.

to pull the {id} or {code} elements from the /v2/suppliers collections you want:

/v2/suppliers/{id}

/v2/suppliers/{code}

Obviously, that creates a huge potential for ambiguity, which makes all things involved Client, server, and API consumer, work much harder to know which supplier the URI references.

To simplify the engine (and the lives of your API consumers), pick one to identify the target resource and use the query pattern for all others. ID seems the better of the two in this case.

Your API sould use:

/v2/suppliers/{id}

/v2/suppliers?code={code}

Reply

Jazz says
October 27, 2019 at 8:14 am

What about really singular document URI:

/v2/supplier/{id}

/v2/suppliers?code=}code}

And then you can also have a qualified collections, that does not mix itself up with single resource identifier:

/v2/suppliers/top

Or maybe better would be

/v2/top-suppliers

?

Reply

The Mikester says
November 27, 2019 at 7:08 pm

We've addressed this use-case by just using the following:

/api/v1/users/{id}

/api/v1/users?filter[code]=code

The first is *the* method of returning a single resource by its primary identifier. This is usually a database primary key in our case.

The second is the method of returning a list of resources, where any number of fields in the resource can be used to filter (there is also sorting and paging implemented as query options as well). In the case where "code" is referring to a single resource, a list of one item is returned.

Paulus says
February 14, 2020 at 6:00 pm

Option 3.

/v2/suppliers/{id}

/v2/suppliers/code/{code}

Reply

Jorge Escobar says
April 2, 2019 at 11:01 am

Great article!, how do you think access roles should be handled? for example we have two types of users (client and manager), a car can be created by each one of them, but the manager can create cars for other clients, and the client only for himself.

We have the identity of the user from the token, so when designing the endpoint we are considering several options:

1. one unique DTO that has the ids for the client and manager

the endpoint would be the same

api/cars/

{

“carName”:””,

“ownerId”:””,

“assignerId”:””

}

2. have different endpoints with different DTOs for each one, something like

api/manager-management/cars

with this DTO

{

"carName":"",

"ownerId":"",

"assignerId":""

}

api/client-management/cars

with this DTO

{

"carName":"",

"ownerId":""

}

3. other….

What do you think would be a good way to go?

Reply

Jorge says
April 29, 2019 at 7:50 pm

Use 1.

Since in the frontend clients shouldn't be able to set the ownerId and managers should, you must know the role for that (send it in the token).

In the backend you can use the role in the token to validate the request, ex. if role is client the ownerid must be equal to assignerId if not the client could be hacking the API request.

best regards.

Reply

Scott says
August 23, 2019 at 8:31 pm

I agree with Jorge. Saying it in a different way:

Your API needs to be able to authorize the user to perform the task. Otherwise, in option 2, anybody could call the manager API and still create cars for others. This could be done by a different client, or by someone maintaining your code (on purpose or by mistake). This is a huge security hole.

Yeah, I know. It means you need to manage authorization code on the client and in the API.

There are numerous options for this.

role-based

resource-based

claims-based

and more

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-2.2

Reply

jmzc says
March 11, 2019 at 7:13 pm

If accountId is an unique global identifier on system , and I want to DELETE an account which is better ?

DELETE /customers/{customerId}/accounts/{accountId}

or

DELETE /customers/accounts/{accountId}

or

DELETE /accounts/{accountId} ?

My question is about if I must provide the customerId identifier (although its not necessary to find the resource ,accountId is enough)

Regards

Reply

Admin says
March 12, 2019 at 9:43 am

To me, `DELETE /accounts/{accountId}` is good. No need to complicate this simple requirement.

Reply

Ptaveen says
March 26, 2019 at 12:16 pm

If thats the case, only use of having customers/{Id}/ as prefix in URI would be while adding a new account.

Reply

Scott says
August 23, 2019 at 8:36 pm

…or getting the accounts for a customer.

or perhaps I don't understand your comment.

Reply

Amark says
February 27, 2019 at 7:48 am

What should I name my api if I want to populate some fields? For example, I would like to get the course with university field and the city field of university populated, what is the best practice to do it? Would GET api/course/populated=[university,university.city] work fine?

Reply

Menuka Ishan says
February 28, 2019 at 8:04 am

You can use JSON object to send data to the API.

Typical Spring boot example will look like below.

As there we use User objects, you can use University Object, where you can include all the details you need.

```
@PostMapping("/users")
    public User create(@RequestBody User user) {
        return userService.create(user);
    }
```

This is how the User class looks like.

You can use implementation like this on your language too.

```
public class User {
    private int id;
    private String firstName;
    private String lastName;
    private String email;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Reply

Jorge says
April 29, 2019 at 7:56 pm

I personally prefer to use the filter _fields in query parameter to limit the fields i want to return from the backend.

Ex.

GET api/courses/populated?_fields=university,university.city&_sort=-university.city

I don’t understand the populated resource, maybe you can remove it.

—-Get all courses only field university and city, order descending by city:

GET api/courses?_fields=university,university.city&_sort=-university.city

Reply

Zero Khor says
February 18, 2019 at 3:49 am

http://api.example.com/song-management/users/{id}/playlists

in php does this mean this?

$_GET[‘song-management’] = ‘users’;

$_GET[{id}] = ‘playlists’;

or it’s actually..

$_GET[‘page’] = ‘song-management’;

$_GET[‘find’] = ‘users’;

$_GET[‘select’] = {id};

$_GET[‘show’] = ‘playlists’;

i'm a bit lost in how the API's URI should behave and how in php would think.

Reply

Sam Butler says
March 9, 2019 at 10:07 am

In PHP the superglobal variable $_GET is used to access query string parameters. If you pass a URL with …/document?foo=bar then you would access the value of 'foo' with $_GET['foo']. For managing URIs in a RESTful API written in PHP I recommend Slim Framework. There you define the routes individually or by using groups to form a hierarchy, including variables like {id} in your example. If you try to assign each part of the URI to an associative array you will lose flexibility. What happens when in your example we go to …/playlists/3/songs/7/history? You have to think of an index name for each part of that, which isn't going to happen. More likely you'll convert the entire URI into a string and use regular expressions to match predefined patterns that refer to a particular controller or controllers. But this would be better handled by a well-maintained framework, and Slim is the best I've found on PHP for lightweight RESTful APIs.

Reply

Phillip says
February 13, 2019 at 10:48 am

Any thoughts on how to handle something like tiers/grades/reliability-levels in the API resource naming? The use case being that the same resource path has the option to go via two different paths. Functionally, both the paths do the same thing, such that hitting either path would have the same result (based on what API is expected to do), but the approach taken is different, which leads to some non functional differences. It is not also an optional thing in a sense that certain set of objects are meant to hit one tier vs the other.

A simple example being a logging API

```
POST /v1/logs
```

whose job is to dump the log into the system. But based on what tier was chosen, the availability/indexing/replication of logs may change.

Should this tier information be incorporated as part of the url or should it be placed somewhere else?

Something like

```
POST /v1/tier1/logs
```

Reply

korkis says
February 8, 2019 at 8:07 am

I think it is convenient to use only Post and Get. I hope DELETE and PUT are depreciated!

Reply

DG says
February 13, 2019 at 8:57 pm

POST are not intended to do what PUT does. A POST should always create a new resource where a PUT replaces an existing one.

You're free to use HTTP however you want, but if you go down that road, it's no longer a REST API. I know if I come across a partner's API that uses POST for every operation I tend to roll my eyes and it gives me the impression that they don't quite know what they're doing.

If I were you, I would reconsider your stance on other HTTP verbs and study their purpose before committing to another non-REST, HTTP based API structure simply for "convenience".

Reply

Sam says
June 14, 2020 at 1:35 am

> A POST should always create a new resource where a PUT replaces an existing one.

There are edge cases. PUT need not result in actual resource creation. POST need not either. The more important invariant here are for PUT the subordinate resource identifier is *already known* by the client and the request is idempotent (meaning no new resources with new identities can ever be created with a PUT). POST is not idempotent and allows for creation of new resources with new identities, with this is not necessary for it's use.

Reply

Zero Khor says
January 29, 2019 at 8:08 am

Hi, I'm new to API.

The example u given using the "/" where it seems to be using "path name" as "argument" or they are actually different directories?

http://api.example.com/device-management/managed-devices/{device-id}

meaning?

api.example.com/index.php

— device-management/index.php

— — managed-devices/index.php

— — — {device-id}/index.php

i got a feeling im getting this wrong. (seriously wrong)

Reply

Admin says
January 31, 2019 at 6:27 am

What I mean is :

http://api.example.com/device-management/managed-devices/

http://api.example.com/device-management/managed-devices/1

http://api.example.com/device-management/managed-devices/2

http://api.example.com/device-management/managed-devices/3

Reply

Ju says
February 10, 2019 at 4:36 pm

Usually you would write one script that catches all the requests. So "/device-management/managed-dives" would be a parameter, not actual folders. Some frameworks use it like "https://example.com/index.php?q=/device-management/managed-dives" unless you set up mod_rewrite. The parameter then gets parsed within your script.

You could also use the https server (Apache, nginx, lighttpd, etc) to split the parameter into different variables and pass them to your php script. But all the common frameworks I've seen (laravel, symfony, codeigniter) actually do in their php scripts for some reasons.

Reply

Zero Khor says
February 15, 2019 at 10:16 am

Thanks.

I was looking at the usage of .htaccess.

still trying to see how i can past all the /var1/var2/var3 into actual $array[0] = "var1";

then the part where i need to specifically do /var3?foo=bar and i would still get $_GET['foo'].

Reply

Florian F says
December 3, 2018 at 10:39 pm

Hi,

You say elsewhere that a resource should have a single logical URI. But the example "/customers/{customerId}/accounts/{accountId}" could also be written "/accounts/{accountId}", given that account IDs should be unique within a bank. So the URI is not unique.

Reply

Admin says
December 6, 2018 at 6:39 am

Very good question. I will argue two things.

1) Account Id and account number are two different things. Account numbers are NPI (sensitive) data so I will never put them in URI/URL. Account id is some generated number (like primary key) for account record.

2) accountIds in both URIs can be different numbers. In `/accounts/{accountId}`, it will be primary key for account record, while in second case it will be primary key which holds the relationship between account and customer. (It is just one example).

Please note I am saying these ids to be primary keys in respective tables just me make things more relatable. An id can be a derived value as well.

Reply

javatec says
November 13, 2018 at 9:56 pm

Excellent article, thank you for your hard work. Could you please provide a suggestion on:

1. if we were to use a sensitive information like insuranceId as a search criteria, should the search be made GET or POST? (since everyone says that providing sensitive information in GET URL is not a good idea)

2. How to handle GET search if we have complicated search criteria like:- searching on a user where first_name starts with a and age is greater than 17 and sort by last_name and show pages 3-7

Reply

Naresh P says
November 28, 2018 at 10:23 am

For your question 1, use HTTP Headers for any sensitive information.

Reply

Frans says
September 12, 2019 at 3:29 pm

Search criteria should be public information, so if you're searching on something that is not public, you might want to take another look at your requirements.

What does sensitive mean? Your entire request should go via HTTPS anyway, so both the URL and the headers and the body would be encrypted. What you might be worried about is access logs where the URLs are logged (so if these are readable by a wider audience than the users of your system who already have access to that information, that might be a problem).

How to handle GET if we have complicated search criteria:

– you could have a min_age query parameter and pass in 18

– you could have an age query paramater and allow values like '>17'

– you could define a whole query language and pass the query in via a 'query' or 'q' parameter (like Google search)

For pagination, check out http://otac0n.com/blog/2012/11/21/range-header-i-choose-you.html which I think is an elegant solution. Alternatively, have your API accept query parameters like the '$top' and '$skip" parameters that OData declares so you can do /accounts?skip=10&top=20 to get results 10 through 30.

Reply

Admin says
November 10, 2018 at 2:13 pm

No. They are not mine. They have been used in other references as well. Though not very widely used.

Reply

Luke says
November 7, 2018 at 4:32 pm

Thanks for the article! Been looking for something just like this! One part still confuses me however:

You recommend above to use the following for collections:

```
http://api.example.com/user-management/users/{id}
http://api.example.com/user-management/users/admin
```

But wouldn't the system just look at /users/admin and think that "admin" is a user ID? This is a problem I've run into before, and is actually what I was researching when I came across this post 🙂 As soon as you allow resources/{variable}, you can no longer put any controller verbs or anything after /resources/ because it'll get interpreted as your variable.

One workaround for this seems to be changing around the order you list your URLs in in your backend, putting the catch-all URL looking for a variable last. But that's always seemed a little hack-y and I don't like that it imposes restrictions on how I might want to organize my URLs in the backend.

Any suggestions for this? Is that workaround in fact standard practice?

Thanks.

Reply

Admin says
November 8, 2018 at 5:21 pm

I had used 3 different examples to represent the document resource. It was never meant to be used collectively. I will also not recommend to use `/{id}` and `/admin` both for same API.

Rather I will go for `/roles/{roleName}` e.g. `/roles/{admin}` which will return all users with admin access.

Reply

Spoilt says
January 23, 2019 at 4:30 pm

To return all users with admin access, isn’t better to use /users?role=admin ?

Reply

Admin says
January 25, 2019 at 4:01 am

That also look OK to me. It is something more of preferences.

Reply

Eirikr says
August 4, 2019 at 6:14 pm

/roles/admin should not return you users with admin access.

Instead it will return a document describing the admin role.

Reply

ju says
February 10, 2019 at 4:47 pm

It's also possible to have only numeric IDs. So the server script could distinguish wether it's an ID or a role/username/whatever. Of course that also imposes some additional limitations like prohibiting numeric and duplicate usernames. Having "/users/{username}" and "/users/{user-id}" wouldn't seem too illogical though (but on the other hand: what's the purpose?). For roles I'd use another way so no one gets confused. You could even use "/users/{numeric-userid}" and "/users/roles/{role-id}" on the same api if you wanted to.

Reply

Eddy says
November 3, 2018 at 6:52 am

Well summarized blog.

I'm curious to find out what you think about designing the user authentication aspects of an application.

I was thinking of following the Controller archetype as follows:

For login:

```
.../user/login
```

For logout:

```
.../users/{id}/logout
```

Reply

Admin says
November 5, 2018 at 6:29 pm

`/login` and `/logout` are good enough for me. No need to put "id" in request path, because it will be stored in session in the server.

Also, I will not put `/user` in the URL until I have a good reason for it.

Reply

Brian Sheldon says
August 19, 2019 at 7:10 pm

Login and Logout are verbs and should not be part of the resource URI.

Instead consider something a like

POST /auth

DELETE /auth

Login by POST-ing your credentials

Logoff by DELETE-ing the auth

If that doesn't offer enough options for you, or to support additional admin resources or other use cases, this can be extended with additional URIs or query params:

/auth?user={user_id}

/auth?action=logout

/auth/{auth_id}

Session history for a user can be made available at:

/user/{user_id}/auth

To see currently active sessions:

/user/{user_id}/auth?active=yes

And the list goes on....

Reply

Josef Sábl says
January 23, 2020 at 2:02 pm

How do you actually logout of the RESTful API? It is supposed to be stateless isn't it.

We have an endpoint /authenticate that requires Http Basic Auth credentials and generates bearer token.

Client then must send the token with each subsequent request.

The token is self contained and contains all the info to authenticate the user as well as limited validity.

User can refresh the token prolonging the validity using the old, soon to expire token.

Reply

Marcel says
October 31, 2018 at 7:58 pm

Very good article, but I have a question,

How could you do if you wanted an API that has the following functions with differents QueryString/ParameterString/Body/Headers in a Messaging App?

Example:

SendMessage

RedirectMessage

RedirectToMessageBlock

ForwardMessage

UpdateAttributesInMessage

All these functions are POST, but If you have a resource name “Message”, you can only have one POST,

http://www.yoursite.com/api/v1/messages

Reply

Mississauga says
November 5, 2018 at 10:45 pm

Marcel,

Here are one way you can handle actions:

http://yoursite.com/api/v1/messages/{id}/send

http://yoursite.com/api/v1/messages/{id}/redirect

http://yoursite.com/api/v1/messages/{id}/redirect-to-message

http://yoursite.com/api/v1/messages/{id}/forward

HTTP POST http://yoursite.com/api/v1/messages/{id}

Enjoy!

JD

Reply

Varunjith Tk says
November 19, 2018 at 11:21 am

send redirect forward etc are verbs. This document says resources should not be verb. So isnt this wrong?

Reply

Brian Sheldon says
August 19, 2019 at 7:32 pm

Avoid uses of verbs in the resource. Also avoid api versioning in the resource.

You can use PATCH to update specific attributes within a message, or PUT to update the entire message.

As for the other options it depends on what your intentions are for each as to what type of solution to use. One option is to POST to the /message/{message_id} with a param such as action=forward as part of the payload.

Reply

Anil Rana says
September 25, 2018 at 6:05 pm

Use "singular" name to denote document resource archetype.

```
http://api.example.com/device-management/managed-devices/{device-id}
http://api.example.com/user-management/users/{id}
http://api.example.com/user-management/users/admin
```

But why plural is used ??

Reply

Admin says
October 3, 2018 at 5:59 pm

Plurals denote collection resources.

Reply

FarzanRNobakht says
August 11, 2019 at 7:35 am

then in the example for the document resource there should be something like:

```
http://api.example.com/user/{user-id}
```

Reply

Oscar says
November 5, 2018 at 7:54 pm

I think the document notation is that

```
http://api.example.com/device-management/...
```

Reply

Benoit Le Nabec says
August 18, 2018 at 12:03 am

About Document archetype. We can do a get/put/delete/patch because we pass an “id”, it’s fine! Is there a situation where you can use a POST on a document. In which case we can we use POST on a Document archetype?

If you see this presentation

https://fr.slideshare.net/domenicdenicola/creating-truly-res-tful-apis

What do you thing about =>page 4 Resource Archetype Document /users/0987/settings. Following you this example would not it be rather relative to controller Archetype ?

What do you thing about => page 5 Archetypye collection : Could you explain the usage of PUT for a collection? It would be in the case where one wishes to replace ALL documents. My understanding A collection is not related to a specific ID. I doesn't understand why the author mention PATCH and DELETE on an archetype collection also? Can you enlighten me on what is "true" of "false"

Reply

SupraDuper says
July 17, 2018 at 11:27 pm

What are the rules for identifying a complex resource by separating a url path? For eg., one of my resource is grouped by 2 tokens. Token one can have many other tokens. The format I am using is:

abc.com/tokens/5678/2345

But there is no page for abc.com/tokens/5678. Should I club "5678/2345" as "5678-2345" or using any other character?

Reply

Admin says

July 18, 2018 at 7:29 am

Yes. I will suggest to append token ids using hyphen.

Reply

Bindu says
July 13, 2018 at 5:04 pm

Sir,

I liked your blog. I would suggest to add few navigation buttons (like next, previous, top, bottom) as to continue the flow of learning and avoid scrolling.

Reply

Lewis says
June 21, 2018 at 2:49 pm

“Do not use trailing forward slash (/) in URIs”

I could not disagree more. Resources are in a tree structure, like a file system. URIs in browsers already behave that way as well.

Trailing forward slash designate a collection, it's like a file system directory. Consider the following:

current uri: http://example.com/test and `` would link to http://example.com/

current uri: http://example.com/test/ and `` would link to http://example.com/test/

Exactly same behavior for a Unix-like file system.

All collections should have a trailing slash, without just relying on the name ending with "s" (though I agree with naming collections using plural).

Reply

Admin says
June 23, 2018 at 6:28 pm

Thanks for sharing your views. But, just like you said, I disagree.

Reply

Robert says
January 5, 2019 at 2:50 am

Thats a very opinionated response.

Lewis couldn't be more precise and logic about the functioning.

Case 1:

When you compose APIs in a navigating through the hierarchy you add the name of the resource without an initial slash because it will specified that your are defining the root path.

Eg. If you define an URI for a resource an compose an HTML based on that that uri refers to the root so will be wrong

Case 2:

Developer define endpoint in his/her application without trailing slash and the wants to compile a full UIR with and ID to get sub-resources why you force to pass and ID with and / obviously this goes in the endpoint and you don't want to define 2 endpoints one for listing and one for the sub-resource.

Reply

Anonymous says
April 6, 2020 at 11:48 pm

The problem with your logic is that collections aren't the only resource that can act as a "directory". All resources could potentially contain an infinite tree structure, so you might as well say all resources should have a trailing slash.

A user could have a list of accounts and a separate URI for its address as you could consider address a separate resource. So you have /users/{userId}/accounts and /users/{userId}/address, therefore /users/{userId} is also a “directory” even though it only represents a singular resource, not a collection.

Reply

RAMPRASAD says
June 12, 2018 at 5:58 am

Hi I want get the data by multiple params (name,age,group)

this case can I use GET method as below

http://www.mywebsite.com/my-project/data/name/age/group

What is the best approach

Reply

Admin says
June 12, 2018 at 2:46 pm

I will suggest to use filters e.g.

```
http://www.mywebsite.com/my-project/data?fields=name,age,group
```

Reply

Steve Hunt says
May 24, 2018 at 11:30 am

Is it RESTful to use URI with query parameters for performing a POST/PATCH to a subset of a collection? For example, if I want to set all the live hosts in the Miami datacenter to status “dead”:

POST /hosts?datacenter=miami&status=live

{ “status”: “dead” }

In the response, I would include all the updated hosts and new status. Some fields (e..g hostname, IP address, etc.) would not be modifiable via this method, which would lead a 4XX status code and appropriate error message.

Reply

Rodrigo says

May 23, 2018 at 3:47 am

I like to use this way of calling custom methods or what you call “controllers”.

https://cloud.google.com/apis/design/custom_methods

Basically when you need to perform a special action over a resource that can’t be represented you add the action after a colon at the end.

For example I have this /groups/{id} resource I want to clone, but there is no such method or similar in HTTP, I could just use POST /groups/{id} with the fetched data of the group I want to clone but If I want to execute that in just one action, so I add /groups/{id}:clone .

As stated in the link, there are some considerations when doing this, like using only POST method, but is more clear this way when are you executing something or not.

Also this maps seamlessly with auth scopes. In this case my scope would be ‘groups:clone’.

Thanks for all this huge documentation, it is really helpful.

Reply

Fernando Xavier says
May 9, 2018 at 9:14 pm

Wich pattern i can use in a model with a composite key?

I have an entity called leaderboard that has a key composed by frontend + packagename, how would my endpoint for get a single leaderboard?

```
GET /leaderboard/{frontend}?packagename={packagename} ?
GET /leaderboard/{frontend}-{packagename} ?
GET /leaderboard/{frontend}/{packagename} ?
```

Reply

Admin says
May 10, 2018 at 7:51 am

1) If a frontend has many packages (in sinle screen) then `GET /leaderboard/{frontend}-{packagename}` does not make sense to me. Same for `/leaderboard/{frontend}/{packagename}`. I would not like to make N API calls to load one screen.

2) `GET /leaderboard/{frontend}?packagename={packagename}` looks more flexible to me because it gives you freedom to fetch 1-N packages in single call.

I am expressing my thoughts purely on assumption of your system. I may be incorrect, so please research.

Reply

Ahmed Alejo says
September 26, 2019 at 5:24 pm

i would prefer

```
/leaderboard/{frontend}/packages/{packagename}
```

Reply

Programie says
April 18, 2018 at 5:48 pm

The article says “Do not use trailing forward slash (/) in URIs”.

So why does https://restfulapi.net/resource-naming redirect to https://restfulapi.net/resource-naming/ (note the trailing slash)?

Reply

Admin says
April 19, 2018 at 8:08 am

Very good observation. But this suggestion does not fit here. Reason is the URL structure. Generally in REST APIs, you pass GET parameters after "?" character. e.g. `https://howtodoinjava.com?page=2` [It is invalid URL as per design of blog].

In blogs, mostly paging parameters are passed as `https://howtodoinjava.com/page/2/` [Valid URL].

So, I say its per design followed in most wordpress blogs. And it is not recommended for REST APIs though its possible.

Reply

Sachin Shah says
March 1, 2018 at 8:17 am

Hi,

I have a requirement where within the hierarchical data one of the resource is a collection instead of single id. What is the best way of modelling it? Example below

Requirement: Give me access control list for specific list of roles. One way of modelling it is to allow only one role id, let the consumer call the API multiple times.

access-management/roles/{id}/acls

However, I want to receive all roles in one request and the response to be then able to return ACLs for all roles. Something along the lines

access-management/roles/{ids}/acls

What is the best way to model the API to adhere to REST and URI guidelines?

Reply

Admin says
March 1, 2018 at 5:02 pm

Though, multiple "ACL"s can belong to single role (as sub-collection), still I see it independent resource different from "Role". So model them accordingly.

I will suggest to create following APIs.

HTTP GET access-management/roles

HTTP GET access-management/roles/{id}

HTTP GET access-management/roles/{id}/acls

HTTP GET access-management/acls

HTTP GET access-management/acls/{id}

So, to reply your query, you can use "access-management/roles" having response like this:

```
<roles>
    <role id="1" uri="/access-management/roles/1">
        <name>admin</name>
        <enabled>true</enabled>
        <acls>
            <acl id="11" uri="/access-management/acls/11">
                <name>ADD</name>
                <enabled>true</enabled>
            </acl>
            <acl id="12" uri="/access-management/acls/12">
                <name>DELETE</name>
                <enabled>true</enabled>
            </acl>
        </acls>
    </role>
    <role id="2" uri="/access-management/roles/2">
        <name>user</name>
        <enabled>true</enabled>
        <acls>
            <acl id="11" uri="/access-management/acls/11">
                <name>ADD</name>
                <enabled>true</enabled>
            </acl>
        </acls>
    </role>
</roles>
```

Here you can iterate all rows and find related ACLs. If you want to fetch full details of any single ACL the follow its `uri` attribute. Feel free to add/remove fields as per design.

Reply

Anonymous says
April 6, 2020 at 11:58 pm

In your API suggestions you list “HTTP GET access-management/roles/{id}/acls” and “access-management/acls” separately. That’s two distinct URI’s for the same collection resource “ACLs”

Reply

Admin says
April 7, 2020 at 8:17 am

Not really. I explained in the first sentence of the comment.

Reply

Kuba says
February 1, 2018 at 1:34 pm

Hi,

Great article, but i have a question.

How can I get the filtered list of managed-devices?

My first idea was something like this:

HTTP POST http://api.example.com/device-management/managed-devices

with my list of device-id in body, but you wrote that this URI is for create.

So what should i do then?

Thanks.

Reply

Admin says
February 2, 2018 at 11:15 am

1) In REST, HTTP POST is more close to “create”. I will suggest to use “HTTP GET” with query parameters.

2) Why somebody will filter a collection by device ids? He will use search functionality (text box) directly.

3) Filtering make sense on other parameters e.g. locations, IP ranges etc. For those usecases, use query parameters. e.g.

HTTP GET http://api.example.com/device-management/managed-devices?states=CA,LS

HTTP GET http://api.example.com/device-management/managed-devices?ip-range=127-0-0-1,127-0-0-10

HTTP GET http://api.example.com/device-management/managed-devices?routes=route1,route2,route3

This approach has benefit that you can use/share these URLs (e.g. bookmarks) again and again over the time.

Reply

Kuba says
February 4, 2018 at 9:15 pm

In my interface I have a list of vehicles names and checkbox next to each of them. When a user chooses few and want to e.g. display them on map, I would like to make only one request for the whole data.

Thanks for the answer, I will pass list of ids in querystring.

Reply

Thieu says
September 22, 2017 at 2:45 pm

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs.

Is it RESTful?

Reply

Admin says
September 25, 2017 at 5:25 pm

Yes, it is.

Reply

Sean says
November 8, 2017 at 6:25 am

checkout is a verb and play are verbs and as you point out at the start it is considered bad practice to use verbs in the URI.

“RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb)”

Also you haven't mentioned which HTTP method should be called for the controller pattern but if it is GET then you are changing state via a method that should be idempotent.

If it is POST then the controller pattern is RPC rather than REST. A RESTful API would allow the retrieval of the "checkout" resource via

"GET http://api.example.com/cart-management/users/{id}/cart/checkout"

which doesn't seem to make sense as a request.

It would be better IMO to include a status attribute in the cart resource and update that.

Reply

Admin says
November 10, 2017 at 11:50 pm

We can put actions in controller resources which are not logically mapped to any of CRUD operations e.g.

```
POST /device-management/{id}/alerts/{id}/resend
```

Above operation does not fall under CRUD.

Reply

Claude Cundiff says
June 7, 2018 at 4:43 am

Would it be fair to say that “resend,” while not a direct CRUD type is actually a “Transaction” if we relate it to the Relational Database World?

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

POST COMMENT

Search Tutorials

ADVERTISEMENTS

ADVERTISEMENTS

Learn REST

What is REST?

ADVERTISEMENTS

References

The dissertation by Roy Thomas Fielding

Uniform Resource Identifier (URI, URL, URN) [RFC 3986]

Internet MediaTypes

Web Application Description Language (WADL)

Meta Links

About

Contact Us

Privacy Policy

Blogs

How To Do In Java