Our guide to our unwritten REST API conventions.

reinhart1010.id – 1 May 2024 • (Updated 1 May 2024)
From
https://reinhart1010.id/blog/2024/05/01/our-unwritten-rest-api-conventions.
Scan the QR Code to view the article on your device or web browser.

> An updated version of this may be available on our digital garden.

Naming things is a difficult thing, according to many developers. But not for someone who graduated in an university that's silently houses great REST API experts.

I'm not joking. According to HackerRank in 2021, BINUS University entered the top charts for Asia-Pacific universities (outside of India) according to their technical skills. And how did the university became the top 2 in REST API design, right after Rajshashi University of Engineering and Technology?

Top Universities in APAC (outside of India) by technical skill					
Rank	Python	Java	JavaScript	SQL	Rest API
1	Rajshahi University of Engineering and Technology Bangladesh	Jahangirnagar University Bangladesh	American International University Bangladesh	University of Dhaka Bangladesh	Rajshahi University of Engineering and Technology Bangladesh
2	University of Peradeniya Sri Lanka	University of Peradeniya Sri Lanka	Sri Lanka Institute of Information Technology Sri Lanka	Telkom University Indonesia	Binus University Indonesia
3	Bangladesh University of Engineering and Technology Bangladesh	Sri Lanka Institute of Information Tecgnology Sri Lanka	Universitas Gadjan Mada Indonesia	Institut Teknologi Bandung Indonesia	University of the Philippines Diliman Philippines
4	University of Sydney Australia	The Hong Kong University of Science and Technology Honh Kong	University of Karachi Pakistan	University of Melbourne Australia	The University of Hong Kong Honh Kong
5	The Hong Kong University of Science and Technology Honh Kong	Shahjalal University of Science and Technology Bangladesh	Binus University Indonesia	University of Sydney Australia	University of Karachi Pakistan

Source: HackerRank, 2021.

1. Use semantic REST naming scheme.

I know, there's no such a standard term for "semantic REST API" like how it did with HTML, but there's also a reason why HTTP requests are divided into GET and POST, which then added by PUT, PATCH, DELETE, OPTIONS, and others.

We commonly use **nested model-centric path approach**, where the REST endpoint paths should be named after the model. No, we don't mean `GET /api/get-embed-from-post`, but as straightforward as `GET /api/posts/embed`. Using the `/posts/` prefix signifies the endpoint belong to the `posts` model. **The model name must be referenced in their plural form.**

We first encountered this model-centric approach in Laravel, where controllers (as in the Model-View-Controller / MVC structure) are mapped into an opinionated set of REST request verbs (e.g. GET, POST) and paths. Taking from Laravel's own example, we decided to define a similar schema:

Verb	URI	Description
GET	/photos	Get an index listing of photos. Get (all) current photos. Can also be potentially used to search for photos.
POST	/photos	Submit a new photo.

Verb	URI	Description
GET	/photos/{id}	Show a specific photo by its (internal) ID.
PUT/PATCH	/photos/{id}	Update the photo information.
DELETE	/photos/{id}	Delete the photo.

You may also notice that some endpoints are not listed here. They are commonly used to display the front-end instead of doing the back-end logic. For example, GET `/photos/{id}/edit` does not edit the actual photo. It only shows a webform to the user, whereas the PUT/PATCH `/photos/{id}` one does the actual logic.

> **Note:** There are some obscure HTTP request verbs listed in https://www.w3.org/Protocols/HTTP/Methods.html that we don't use, such as CHECKIN, LINK, and TEXTSEARCH. These are rarely recognized in HTTP request and server libraries, so they may just add an additional burden for us and others to implement.

2. Use kebab-case for paths, snake_case for parameters.

Another common practice that we used here is to use `kebab-case` for REST paths and `snake_case` for parameters, even though that those are defined in the JSON format (which commonly written in `camelCase` instead). Yes, we do represent **data model types** (i.e. the kind of data being represented in objects, structs, and database tables) in `PascalCase`, but when it became part of the REST request, we use the former ones.

> You can learn more about the differences between these four naming conventions in the FreeCodeCamp.

The reason why we used such conventions is to tolerate case-sensitivity, so user do not have to use certain uppercase letters to refer to the right data model. Additionally, web services conventionally used the `snake_case` for request parameters, so we leave the convention as-is for compatibility purposes.

3. Every response data should correspond closely to the enquired model.

Well, we said we're grouping things based on its model, right? So there has to be a standard way to represent the model and its items for each model-related responses.

And that's what we actually did: **always map every model across database, source codes, and input/output serialization.**

- In database, this means the table columns.
- In the source code, this is most likely to be objects (as in OOP) or structs as a fallback.
- In data serialization, we can use common serialization objects including JSON and YAML. We still do prefer JSON for REST API responses.

For example, creating a new item under the model should give a HTTP response containing the new model, instead of simply signalling that “the insertion is complete” or just the item IDs. Similarly, when updating an item or two, the response should include the updated model items. There might be some exceptions for this, such as when doing batch processing, we could reduce the unnecessary response data load by signifying the updated item IDs, including which one were unchanged or faulty.

Tabel 3.36 Daftar Tipe Data yang Digunakan Penulis dalam Mengimplementasikan Para *Entity* dalam Sistem Aplikasi AEP Mobile

Tipe Data	MySQL / MariaDB	Dart	PHP
bool	UNSIGNED TINYINT(1)	bool	bool
integer	INT	int	int
unsigned integer	UNSIGNED INT	unsigned int	unsigned int
bigint	BIGINT	BigInt	bigint
unsigned bigint	UNSIGNED BIGINT	BigInt	unsigned bigint
float[a]	FLOAT	double	float
unsigned float[a]	UNSIGNED FLOAT	double	float
double[a]	DOUBLE	double	float
unsigned double[a]	UNSIGNED DOUBLE	double	float
string	• CHAR untuk *string* dengan panjang terbatas • VARCHAR untuk *string* dengan panjang beragam dan terbatas • TEXT dan LONGTEXT untuk *string* dengan ukuran lebih panjang daripada CHAR dan VARCHAR	String	string
dynamic (menerima tipe data apapun)	-	dynamic	mixed
generic (tipe data placeholder untuk pengolahan data pada class dan method)	-	Ditaruh di dalam kurung siku Contoh: <U>, <K, V>	Ditaruh di dalam kurung siku dan dibubuhkan kata kunci *@template* pada PHPDoc **Contoh:** <U>, <K, V> Contoh 2: /** * @template T */

In his own thesis, Reinhart listed his data type mapping across MySQL, MariaDB, Dart, PHP to ensure that the app's entity models are properly represented from the database to the backend and frontend.

4. Our standard REST response.

```
{
  "status": "OK",
  "data": /* ... */
  "warning": /* ... */
}
```

```
{
  "status": "KO",
  "error": /* ... */
  "data": /* ... */
}
```

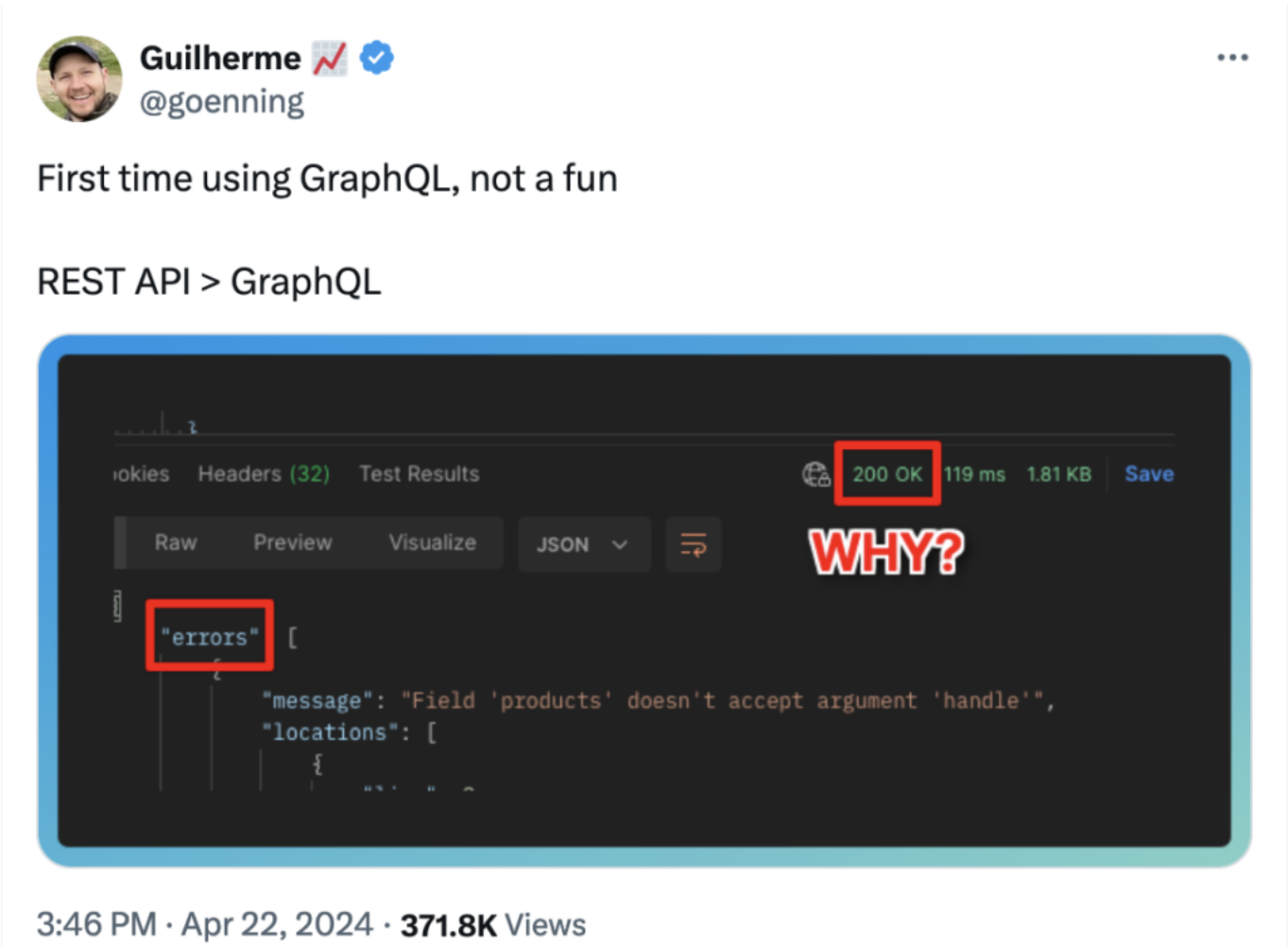
Our standard response is always written in JSON unless a different format is required, e.g. returning an Atom/RSS feed in XML. This response is heavily inspired by the simplistic DuckDNS API and other REST API designs.

We commonly fill in the `error` parameter with error codes, which may vary from system to system. Some of the common ones include `{{MODEL}}_NOT_FOUND` accompanied with the HTTP 404 status code and perhaps some additional diagnostic data located under the `data` paraeter. But some systems which require further debugging may return the error message as-is from the related software libraries on that `error` parameter.

5. Do not afraid of using HTTP response codes.

```
HTTP 200
{
  "statusCode": 404,
  "data": "Blog Post not found."
}
```

The enterprise developer who made this should go to the hell!



Source: https://twitter.com/goenning/status/1782330200958615637

Don't worry, angry developers. We are still using HTTP status codes responsibly!

6. Now, what if we really need an endpoint that does not really fit into the above conventions?

We have an internal term for it: **import/export endpoints**. They are endpoints which processes a standard model representation from or to the non-standard ones, such as importing items from CSV, or outputting a list of posts in Atom/RSS format.

These endpoints do not follow some of our path conventions, but still need to contained inside a model path, such as `/posts/feed` and `/products/import`. The expected input and output data formats can deviate from the standard ones, too.

Some of our systems also have to deal with other REST API standards, such as utilizing the `/.well-known/` path for things related to digital identity and website verification. Many also respect the classic `/ping` endpoint, which expects to return a plaintext response of `pong`, to check whether the server is still active.

And lastly, we also have a dedicated endpoint group, name `/test`. Since our model names are plural, we do not really care if the name conflicts with the model `Test`, since the model will eventually be represented as `/tests/`. These endpoints are meant for utilities to help developers integrate to our systems, such as `/test/access-token` to check whether an access token is still valid, and

`/v4/test/ping` (alongside `/ping`) to check whether the service supports the REST API Version 4 schema.