brijpandeyji

REST API Design Best Practices

The Ultimate Guide to Building Effective APIs

swipe >

General Concepts

KISS

Anyone should be able to use your API without having to refer to the documentation.

- Use standard, concrete and shared terms, not your specific business terms or acronyms.
- Never allow application developers to do things more than one way.
- Design your API for your clients (Application Developers), not for your data.
- Target major uses cases first, deal with exceptions later.

Example

GET /salaries, GET /employees, GET /departments, ...

CURL

You should use CURL to share examples, which can be easily copy/paste.

Example

```
CURL -X POST \
-H "Accept: application/json" \
-H "Authorization: Bearer at-9085000-19a8-46a2-908e-33d4057128e7" \
-d '{"state":"running"}' \
https://api.company.com/v1/employee/11/orders?employee_id=111
```

Medium Grained Resources

You should use medium grained, not fine nor coarse. Resources shouldn't be nested more than two level deep :

```
GET /employee/11
```

```
{
   "id": "11",
   "firstname" : "John",
   "name" : "Brown",
   "address" : {
      "street" : "425 Buckland",
      "country" : {"name" : "USA"}
   }
}
```

You may consider the following five subdomains

- Production - https://api.website.com
- Tests - https://api.sandbox.website.com
- Developer Portal - https://developers.website.com
- Production - https://oauth2.website.com
- Tests - https://oauth2.sandbox.website.com

Security : OAuth2 & HTTPS

You can use OAuth2 or JWT to manage Authorization.

- OAuth2 & JWT matches 99% of requirements and client typologies, don´t reinvent the wheel, you´ll fail.

swipe >

URLs

Nouns

You should use nouns, not verbs (vs SOAP-RPC).

`GET /employees` not `/getAllEmployees`

Plurals

You should use plural nouns not singular nouns to manage two different types of resources :

- Collection resource : /employees
- Instance resource : /employees/11

You should remain consistent.

`GET /employees/11` not `GET /employee/11`

Consistent Case

You may choose between snake_case or camelCase for attributes and parameters, but you should remain consistent.

Example

GET /salaries?id_user=11 or GET /salaries?idUser=11

POST/salaries {"id_user" : "11"} or POST/salaries {"idUser" : "11"}

If you have to use more than one word in URL, you should use spinal-case (some servers ignore case).

POST /specific-salaries

Versioning

You should make versioning mandatory in the URL at the highest scope (major versions). You may support at most two versions at the same Tme (Native apps need a longer cycle).

GET /v1/salaries

Hierarchical Structure

You should leverage the hierarchical nature of the URL to imply structure (aggregation or composition). Ex : an order contains products.

Example

GET /salaries/3456/departments/1

CRUD-like Operations

Use HTTP verbs for CRUD operations (Create/Read/Update/Delete).

HTTP Verb	Collection : /orders	Instance : /orders/{id}
GET	Read a list orders. 200 OK.	Read the detail of a single order. 200 OK.
POST	Create a new order. 201 Created.	-
PUT	-	Full Update : 200 OK./ Create a specific order : 201 Created.
PATCH	-	Partial Update. 200 OK.
DELETE	-	Delete order. 204 OK.

POST is used to create an instance of a collection. The ID isn't provided, and the new resource location is returned in the "Location" Header.

POST /salaries {"state" : "processing", "id_user" : "11"}

201 Created

Location: https://api.website.com/salaries/3456

But remember that, if the ID is specified by the client, PUT is used for Create.

PUT /salaries/3456

201 Created

PUT is used for Update to perform a full replacement.

PUT /salaries/3456 {"state" : "credited", "id_user" : "11"}

200 Ok

PATCH is commonly used for partial Update.

PATCH /salaries/3456 {"state" : "credited"}

200 Ok

GET is used to Read a collection.

```
GET /salaries
200 Ok
[ {"id":"3456", "state":"credited"}
   {"id":"9874", "state":"processing"}]
```

GET is used to Read an instance.

```
GET /salaries/3456
200 Ok
{"id":"3456", "state":"credited"}
```

brijpandeyji

Query Strings

Search

You may use the "Google way" to perform a global search on multiple resources.

Example

```
GET /search?q=processing+credited
```

Filters

You should use '?' to filter resources

```
GET /salaries?state=credited&id_user=11
```

or (multiple URIs may refer to the same resource)

```
GET /salaries/11/salaries?state=credited
```

swipe >

brijpandeyji

Pagination

You may use a range query parameter. Pagination is mandatory : a default pagination has to be defined, for example : range=0-25
The response should contains the following headers : Link, Content-Range, Accept-Range.

Note that pagination may cause some unexpected behavior if many resources are added.

/orders?range=18-25

206 Partial Content

Content-Range: 18-25/971

Accept-Range: order 10

Link : https://api.website.com/v1/salaries?range=0-11; rel="first",

https://api.website.com/v1/salaries?range=28-53; rel="prev",

https://api.website.com/v1/salaries?range=57-69; rel="next",

https://api.website.com/v1/salaries?range=71-813; rel="last"

swipe >

Partial Responses

You should use partial responses so developers can select information needed and optimize bandwidth (essential for mobile development).

```
GET /employees/11?fields=firstname,name,address(street)

200 OK

{  "id" : "11",
   "firstname" : "John",
   "name" : "Brown",
   address : {"street":"425 Buckland "}
}
```

Sort

Use ?sort=attribute1, attributeN to sort resources. By default resources are sorted in ascending order. Use ?desc=attribute1, atributeN to sort resources in descending order.

```
GET /companies?sort=rating,reviews,name;desc=rate,reviews
```

swipe >

brijpandeyji

URL reserved words : first, last, count

Use /first to get the 1st element

```
GET /employees/first
200 OK
{"id":"3456", "state":"credited"}
```

Use /last to retrieve the latest resource of a collection

```
GET /employees/last
200 OK
{"id":"6789", "state":"processing"}
```

Use /count to get the current size of a collection

```
GET /employees/count
200 OK
{"2"}
```

Other Key Concepts

Content Negotiation

Content negotiation is managed only in a pure RESTful way. The client asks for the required content, in the Accept Header, in order of preference. Default format is JSON.

Example

Accept: application/json, text/plain not /employees.json

I18N

Use ISO 8601 standard for Date/Time/Timestamp : 1978-05-10T06:06:06+00:00 or 1978-05-10 Add support for different languages.

Accept-Language: fr-ca, fr-fr not ?language=fr

swipe >


brijpandeyji

Cross-origin Requests

Use CORS standard to support REST API requests from browsers (js SPA...). But if you plane to support Internet Explorer 7/8 or 9, you shall consider specifics endpoints to add a Jsonp support.

- All requests will be sent with a GET method!
- Content negotiation cannot be handled with Accept Header in Jsonp.
- Payload cannot be used to send data.

POST /salaries and /salaries.jsonp?method=POST&callback=foo

GET /salaries and /salaries.jsonp?callback=foo

GET /salaries/1234 and /salaries/1234.jsonp?callback=foo

PUT /salaries/1234 and /salaries/1234.jsonp?method=PUT&callback=foo

Warning : a web crawler could easily damage your applicaTon with a method parameter. Make sure that an OAuth2 access_token is required, and an OAuth2 client_id as well. You can also use other authentication methods.

swipe >

HATEOAS

Your API should propose Hypermedia links in order to be completely discoverable. But keep in mind that a majority of users wont probably use those hyperlinks (for now), and will read the API documentation and copy/paste call examples.

So, each call to the API should return in the Link Header every possible state of the application from the current state, plus self.

You may use RFC5988 Link notation to implement HATEOAS :

GET /employees/11

< 200 Ok

< { "id" : "11", "firstname" : "John", ...}

< Link : https://api.website.com/v1/salaries; rel="self"; method:"GET",

https://api.website.com/v1/addresses/42; rel="addresses"; method:"GET",

https://api.website.com/v1/employees/3456; rel="orders"; method:"GET"

"Non Resources" Scenarios

In a few use cases we have to consider operations or services rather than resources. You may use a POST request with a verb at the end of the URI.

POST /cellphone/91/send

POST /calculate/sum [212,314,6,7,8,91]

POST /convert?from=USD&to=INR&amount=1200

brijpandeyji

For More Interesting Content

Brij Kishore Pandey

Follow Me On LinkedIn

https://www.linkedin.com/in/brijpandeyji/