REST API Design Best Practices

What is a REST API?

- A REST API is an application programming interface that conforms to specific architectural constraints, like stateless communication and cacheable data.
- It is not a protocol or standard.
- While REST APIs can be accessed through a number of communication protocols, most commonly, they are called over HTTPS.
- While REST APIs can use multiple formats for data transfer, most commonly JSON is used.

Think Nouns

```
/getPostById?id=xx

/getAddressesForCustomer?customerId=xx

/createOrder

/updateOrder?id=xx
```

```
GET /posts/xx

GET /customers/xx/addresses

POST /orders

PUT /orders/xx
```

* in Plural

Only 4 Verbs

Create - POST

Read - GET

Update - PUT / PATCH

Delete - DELETE

PUT Vs PATCH

GET - to retrieve data

User Model:

```
{ "username": "skwee357", "email": "skwee357@domain.com" }
```

POST:

POST- create an data

```
POST /users

{
    "username": "skwee357",
    "email": "skwee357@domain.com"
}
```

PUT: check if data exist and update

PUT:

```
PUT /users/1
{
    "username": "skwee357",
    "email": "skwee357@gmail.com"       // new email address
}
```

PATCH:

PATCH : always update the data

```
PATCH /users/1
{
    "email": "skwee357@gmail.com"       // new email address
}
```

HTTP status codes in responses

- `200` for general success
- `201` for successful creation
- `400` for bad requests from the client
- `401` for unauthorized requests
- `403` for missing permissions
- `404` for missing resources
- `429` for too many requests
- `5xx` for internal errors (these should be avoided at all costs)

Representing Relationships - Nesting and Embedding

- `GET /tickets/12/messages` - Retrieves list of messages for ticket #12
- `GET /tickets/12/messages/5` - Retrieves message #5 for ticket #12
- `POST /tickets/12/messages` - Creates a new message in ticket #12
- `PUT /tickets/12/messages/5` - Updates message #5 for ticket #12
- `PATCH /tickets/12/messages/5` - Partially updates message #5 for ticket #12
- `DELETE /tickets/12/messages/5` - Deletes message #5 for ticket #12

Questions:

1. What if there is an “owns” relationship between tickets and messages (messages can only exists within tickets)?
2. What if messages can exist independently of tickets?
3. What if messages are frequently requested with tickets?

Allow filtering, sorting, and pagination

Server-side filtering and pagination is important if data is too large.

```
/employees?lastName=Smith&age=30
```

We get:

```
[
    {
        "firstName": "John",
        "lastName": "Smith",
        "age": 30
    }
]
```

How github do pagination:

`https://api.github.com/user/repos?page=3&per_page=100`

Generic Example:

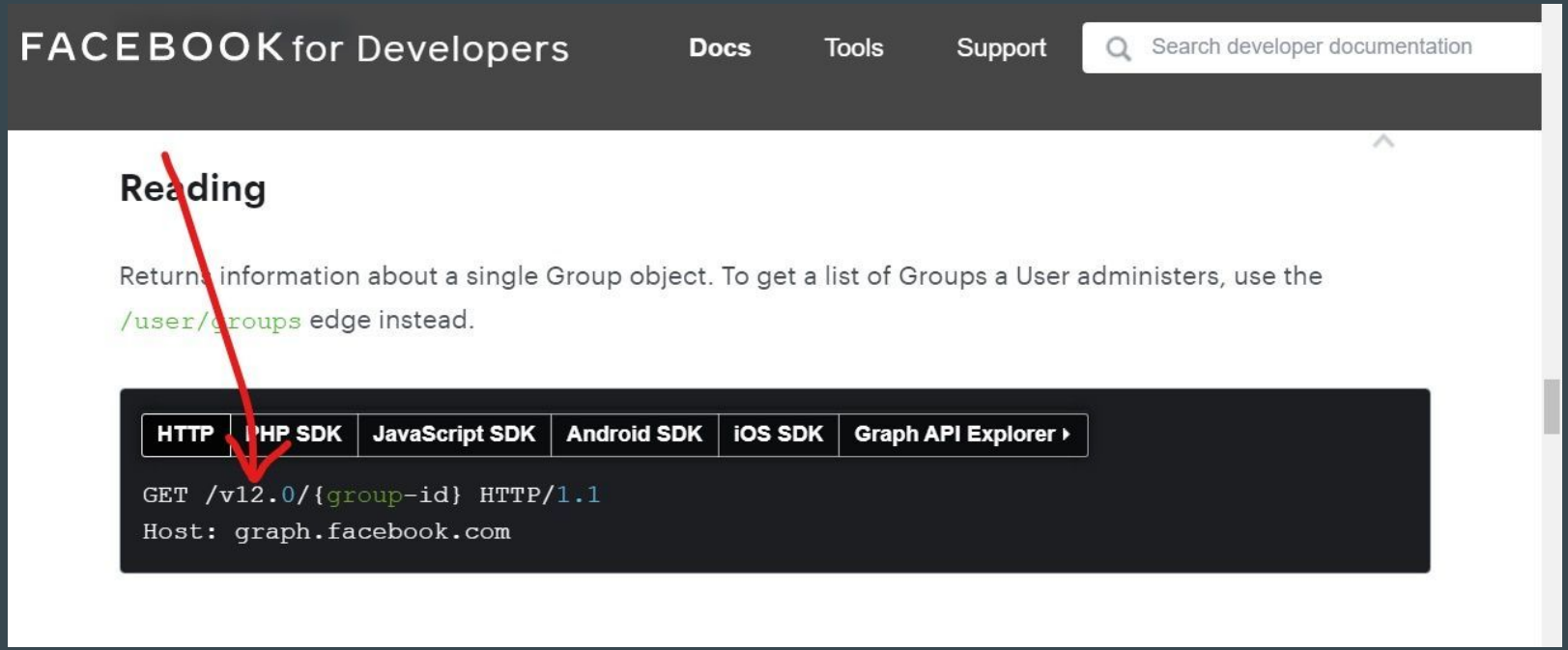
```
// Request => GET /users?page_number=1&page_size=15

// Response <= 200 OK
{
    "page_number": 1,
    "page_size": 15,
    "count": 378,
    "data": [
        // resources
    ],
    "total_pages": 26,
    "has_previous_page": true,
    "has_next_page": true
}
```

Versioning

`https://mysite.com/v1/` for version 1

`https://mysite.com/v2` for version 2



Spotify Example - Versioning and Pagination

Spotify API:

Pagination

Some endpoints support a way of paging the dataset, taking an offset and limit as query parameters:

```
$ curl
https://api.spotify.com/v1/artists/1vCWHaC5f2uS3yhpwWbIA6/albums?
album_type=SINGLE&offset=20&limit=10
```

In this example, in a list of 50 (`total`) singles by the specified artist : From the twentieth (`offset`) single, retrieve the next 10 (`limit`) singles.

Note: The offset numbering is zero-based. Omitting the `offset` parameter returns the first X elements. Check the documentation for the specific endpoint and verify the default `limit` value. Requests that return an array of

Use ISO 8601 UTC dates

Displaying dates in a specific time zone is generally a concern of client applications.

```
{
    "published_at": "2022-03-03T21:59:08Z"
}
```

Provide a health check endpoint

Provide an endpoint (for example `GET /health`) that determines whether or not a service is healthy. This endpoint can be called by other applications such as load balancers to act in case of a service outage.

Return created resources upon POST (And PUT/PATCH)

Because the returned, created resource will reflect the current state of the underlying data source, along with the generated ID. API consumer need not have to hit the API again for an updated representation

```
// Request: POST /users
{
    "email": "jdoe@averagecompany.com",
    "name": "John Doe"
}

// Response
{
    "id": "T9hoBuuTL4",
    "email": "jdoe@averagecompany.com",
    "name": "John Doe"
}
```

Rate Limiting

To prevent abuse, it is standard practice to add some sort of rate limiting to an API. RFC 6585 introduced a HTTP status code 429 (Too Many Requests) for this.

However, it can be very useful to notify the consumer of their limits before they actually hit it. Some popular conventions using HTTP response headers:

- `X-Rate-Limit-Limit` - The number of allowed requests in the current period
- `X-Rate-Limit-Remaining` - The number of remaining requests in the current period
- `X-Rate-Limit-Reset` - The number of seconds left in the current period

Meaningful error codes and messages

Option 1	Option 2
name_too_long	invalid_name
token_revoked	invalid_auth
not_an_admin_user	access_denied
missing_user_id	bad_input

Good errors vs not so good

```
{
    "ok": false,
    "error": "method_deprecated",
    "response_metadata": {
        "messages": [
            "[ERROR] This method is retired and can no longer be used. Please use conversations.list or users.
                conversations instead. Learn more: https://api.slack.com/changelog/
                2020-01-deprecating-antecedents-to-the-conversations-api."
        ]
    }
}
```

Thanks

Q&A

@aj7tt