Rest API Guidelines

- Keep your base URL simple and intuitive.
- A key litmus test is that there should be only 2 base URLs per resource.
- The first URL is for a collection; the second is for a specific element in the collection. `/dogs` `/dogs/1234`
- Keep verbs out of your base URLs. `/dogs` not `/getAllDogs`
- Use `POST`, `GET`, `PUT` and `DELETE` for `CRUD` operations.
- It reads more easily and is more intuitive to use plural nouns.
- Above all, avoid a mixed model in which you use singular for some resources, plural for others. Being consistent.
- Concrete names are better than abstract ones. Example: `/vedios` `/images` not `/assets`.
- Sticking to the max 2 base URL rule, sweep complex relations between resources behind the `?`. Example: `/dogs?owner=johndoe&color=black` not `/owners/1234/dogs/black`.
- Once you have the primary key for one level, you usually don't need to include the levels above because you've already got your specific object. Example: `resource/id?query` not `resource/id/resource`.
- Use HTTP status codes and try to map them cleanly to relevant standard-based code.
- Try to stick to the most relevant and known status code.
- There are over 70 status code and no one will memorize all of them. Using not common codes forces

the client of your API to leave the code and try to search for the meaning of that code.

- There are really only 3 outcomes in the interaction between an app and an API:
  - Everything worked - success
  - The application did something wrong - client error
  - The API did something wrong - server error
- You should not need to go beyond 8 codes:
  - 200 - OK
  - 201 - Created
  - 304 - Not Modified
  - 400 - Bad Request
  - 401 - Unauthorized (authentication error)
  - 403 - Forbidden (authorization error)
  - 404 - Not Found
  - 500 - Internal Server Error
  - note: 404 can be used instead of 401 or 403 for security concerns (To avoid revealing the presence or absence of the resource).
- Make messages returned in the payload as verbose and plain as possible.
- It is desirable to include links for more information in the payload if needed.
- Never release an API without a version and make the version mandatory.
- Specify the version with a `v` prefix. Move it all the way to the left in the URL so that it has the highest scope example: `/v1/dogs`.
- Use a simple ordinal number. Don't use the dot notation like v1.2 because it implies a granularity of versioning that doesn't work well with APIs. It's an interface not an implementation.

- How many versions should you maintain? Maintain at least one version back.
- For how long should you maintain a version? Give developers at least one cycle to react before obsoleting a version.
- Versioning can also be achieved through custom headers or Media Type versioning.
- URL versioning is the most common and more suitable for small API's with limited number of resources.
- Support partial response by adding optional fields in a comma delimited list => `/dogs?fields=name,color,owner`
- Use limit and offset to make it easy for developers to paginate objects => `/dogs?offest=50&limit=25`
- Use verbs not nouns for cases that does not involve returning a resource => `/convert?from=egp&to=usd&amount=5000`
- Support different formats. JSON is the most common default format.
- Follow JavaScript conventions for naming attributes on the response. That is the camel case.
- For searching:
  - Simple search regarding one resource: `/dogs?q=red`
  - Global search: `/search?q=red`
  - Scoped search: `/owners/1234/dogs?q=red`
  - Formatted search: `/search.xml?q=red`
- Facebook provides two APIs (`graph.facebook.com` & `api.facebook.com`) but for the best interest of app developer consolidate API requests under one API subdomain.
- You can have developer portals `developer.domain` and optionally you can provide redirection if the request comes from the browser => `api.domain` redirects

to `developer.domain` when comes from the request made from a browser.

- When client can't handle exceptions, use `suppress_response_code=true` and push any response code down into the response message. This way the response will always be 200 OK.
- When a client supports limited HTTP methods, use `GET` with optional parameter specifying the method => `/dogs?method=post`.
- For authentication, `OAuth 2.0` is recommended.
- Complement your API with code libraries and a software development kit (SDK) to help clients overcome domain knowledge if needed.
- Use the `façade` pattern when you want to provide a simple interface to a complex subsystem. Subsystems often get more complex as they evolve. => `Ideal Design | Facade | Systems`.

Resources: [API Design Book]