REST API

REpresentational State Transfer (REST) is an architectural style that defines a set of constraints to be used for created web services.

1. Way of accessing web services in a simple and flexible way without having any processing
2. Uses less bandwidth
3. Simple, light-weight, highly scalable and maintainable
4. Communication is done via HTTP requests

Client sends a request to the server which can be GET, POST, PUT, PATCH or DELETE (other methods like OPTIONS and HEAD are rarely used)

Server responds back with resources which can be XML, HTML, Image or JSON (JSON is mostly popular)

POST	Create a resource
GET	Get a resource
PUT	Replace the resource with the one being sent / create a resource
PATCH	Update the resource / create a resource
DELETE	Delete a resource

Idempotence: An HTTP method is idempotent when the result obtained is the same regardless of the number of times it is executed. For example, A = 4, this statement no matter how many times you execute, the result won't change but, A++, is not an idempotent request

Architectural Constraints of RESTful API

1. **Uniform Interface**: There should be a uniform way of interacting with the server, irrespective of the device (mobile or web). There are four guidelines of Uniform Interface:
    a. Resource-based
    b. Manipulation of resources through representations
    c. Self descriptive messages
    d. HATEOAS (Hypermedia As The Engine Of Application State)
2. **Stateless**: Server should not store anything related to the session. All the necessary information required to execute a API call should come from the client
3. **Cacheable**: Every response should include whether the response is cacheable or not. Improves performance and availability
4. **Client-Server**: For a client, the server should be kinda black box
5. **Layered System**: Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches
6. Code on Demand: Optional feature, server can provide executable code to the client

If a service violates any of the above constraints, it cannot be called RESTful.

DO'S

1. Keep base URL simple and intuitive
2. There should be only 2 base URLs per resource
3. Use HTTP verbs to operate on collections and elements (POST, GET, PUT, DELETE)
4. Design the API in such a way that developers probably don't need to look at the documentation
5. Use plural nouns for resources
6. Use concrete names rather than abstract names. For example, /media is an abstract name while /images /videos /gifs are concrete names and describe a lot more of what the resource is like
7. Simplify associations between resources and use attributes with the HTTP question mark (?). For example,

   GET /dogs?color=black&state=running&location=park

8. Use only 7-8 HTTP status codes to describe error, not more than 10
9. Make error messages in payload as verbose as possible
10. Always release an API with a version
11. Versions can be specified as /v1/dogs.
12. If response is huge, provide partial response or paginated response
13. Developers should be able to add only required fields in the call
14. In case of pagination, use limit and offset. For example, a response consists of 4000 objects, so the developer can specify the number of objects requires (limit) and the position from which these object start (offset)
15. Make sure to add default values in limit and offset
16. API calls that don't actually work on a resource, for example, calculating taxes or converting currencies, for such endpoints, verbs can be used. For example, /convert?from=USD&to=INR
17. API should support multiple formats, json, xml etc.
18. Follow javascript conventions for naming attributes and use CamelCase
19. Consolidate API requests in one subdomain, for example, developers.facebook.com
20. For exception handling, there should be an options for suppressing response codes (converting all HTTP response code to 200), for example, /dogs?suppress_response_code=true. In such situation, the actual error HTTP code can move to the payload as response_code
21. Use OAuth2.0 for authentication

DON'TS

1. Don't build a chatty API
2. Don't use verbs in endpoints, for example, /getAllDogsWhoAreHungry
3. Don't create too many endpoints
4. Don't be and a**hole in general when building an API