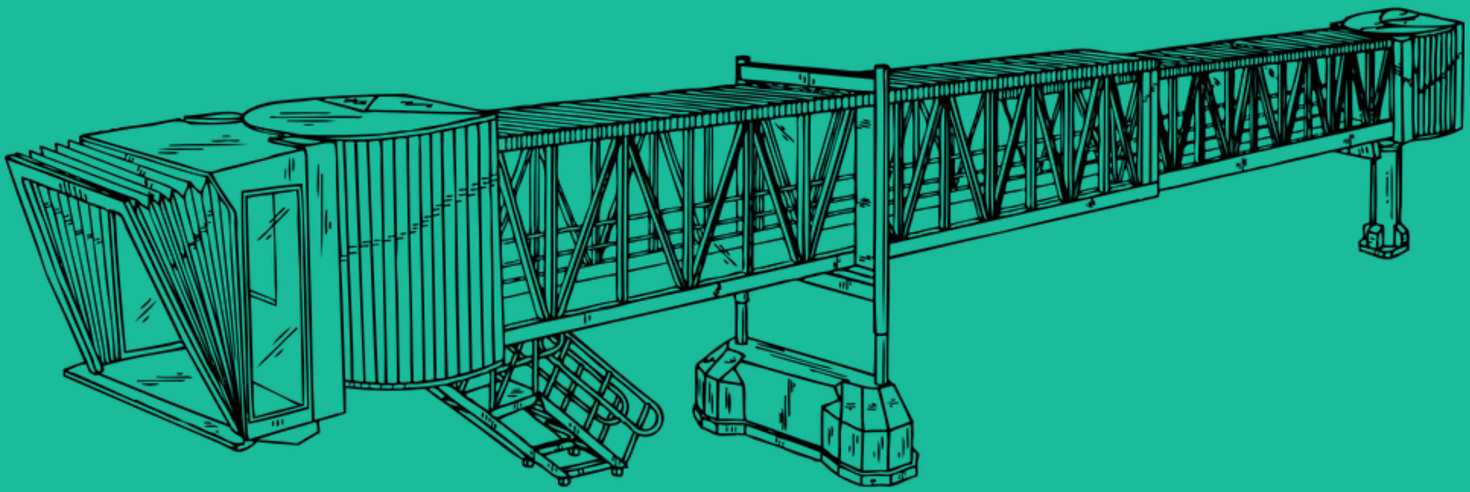
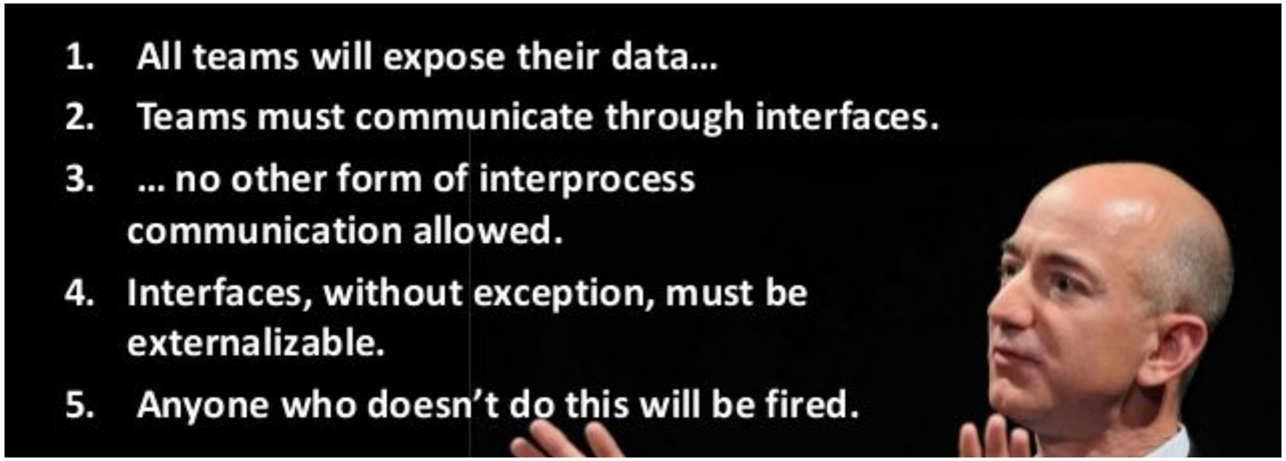


Chapter X-1. API Design

Bilkent University | CS443 | 2021, Spring | Dr. Orçun Dayıbaş

What is API?

- **Definition**
  - Application Programming Interface (defines interactions between multiple software)
  - It defines the kinds of calls or requests that can be made
  - At some point or another, most large companies have built APIs for their customers, or for internal use.
- **Why it is important?**
  - Amazon (2002) / Bezos's API Mendate



1. All teams will expose their data...
2. Teams must communicate through interfaces.
3. ... no other form of interprocess communication allowed.
4. Interfaces, without exception, must be externalizable.
5. Anyone who doesn't do this will be fired.

What is API?

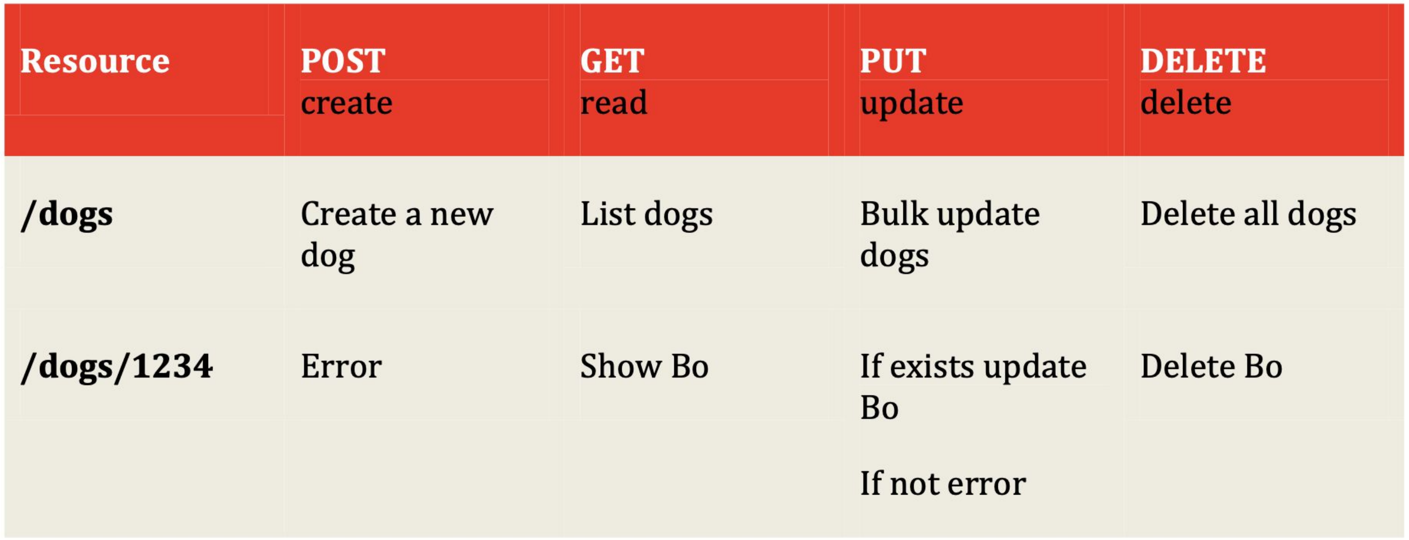
- **Resource-centric vs. behavior-centric**
    - There are two dominant styles for APIs
    - Entity/resource-centric: the client manipulates the entities you expose
        - Entities are in front of the behavior
        - POST /dogs, GET <dog url>, etc.
        - Ex: REST
    - Procedure/Behavior-centric: the client calls your methods/functions
        - Entities are behind the procedures
        - /getAllDogs, /verifyLocation, /checkHealth
        - Ex: RPC

What is a RESTful API?

- **REST is a set of architectural constraints**
  - It is not a protocol or a standard. API developers can implement REST in a variety of ways.
  - Six constraints
    - Client-server
    - Cacheable
    - Stateless
    - Layered
    - Code on demand (optional
    - Uniform Interface
      - Resource identification
      - Resource manipulation with representations
      - Self-descriptive
      - HATEOAS

Nouns are good, verbs are bad

- **2 base URLs per resources**
  - /dogs → collection
  - /dogs/{id} → specific element in the collection
- **Keep verbs out of your URLs**
  - /getAllDogs
  - /newDog
- **Use HTTP Verbs (Only!)**



Resource	POST create	GET read	PUT update	DELETE delete
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists update Bo If not error	Delete Bo

How to pick nouns for our URLs?

- **Plural nouns and concrete names**
    - Above all, avoid a mixed model (be consistent).
    - De facto way; using plural names (more intuitive).
    - “Enough” level of abstraction is needed.
        - /items, /assets → loses opportunity to paint a tangible picture for developers.
        - Instead use /videos, /news, etc.
    - The level of abstraction depends on your scenario.
    - You also want to expose a **manageable number of resources**.
        - Aim for concrete naming and to keep the number of resources less than 20.

Associations

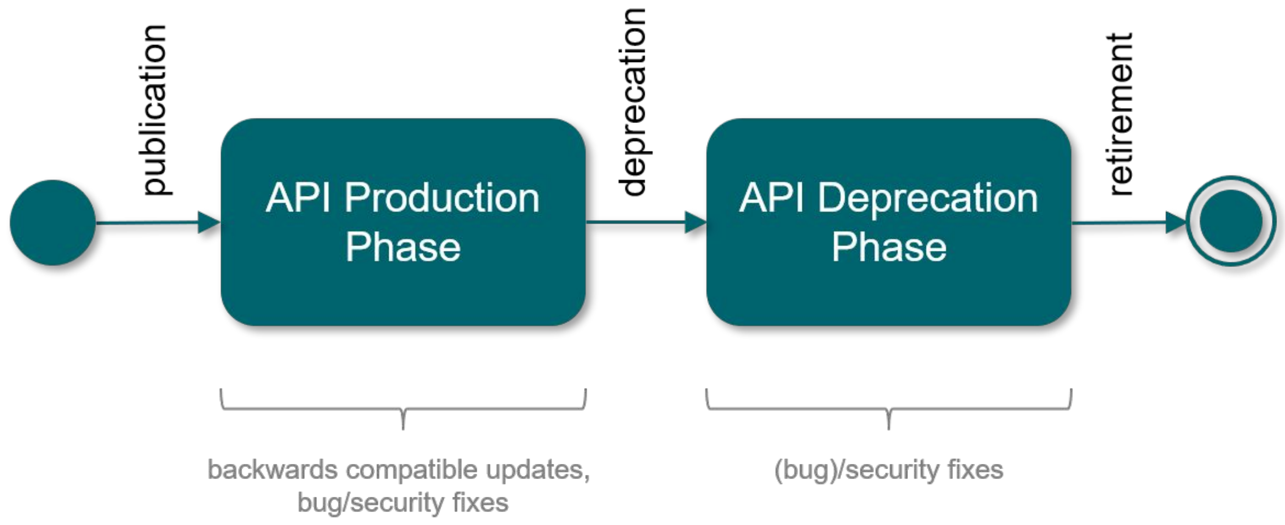
- **A simple way to express relationships?**
  - To get all the dogs belonging to a specific owner, or to create a new dog for that owner, do a `GET` or a `POST`:
    - `GET /owners/5678/dogs`
    - `POST /owners/5678/dogs`
  - The relationships can be complex but once you have the primary key for one level, you usually don't need to include the levels above (you've already got your specific object). In other words, you shouldn't need too many cases where a URL is deeper than what we have above
    `/resource/identifier/resource`
  - Sweep complexity behind the "?"
    - `GET /dogs?color=red&state=running&location=park`

Handling Errors

- **Use HTTP Status Codes**
  - There are over 70 HTTP status codes.
  - Thre most API providers use a small subset. For example, the Google GData API uses only 10 status codes; Netflix uses 9.
    - **Google GData:** 200 201 304 400 401 403 404 409 410 500
    - **Netflix:** 200 201 304 400 401 403 404 412 500
  - How many status codes should you use for your API?
    - Everything worked as expected → Success (200)
    - Caller did something wrong → Client error (400)
    - Callee did something wrong → Server error (500)
  - Make messages returned in the payload as verbose as possible (Code for code, message for people).

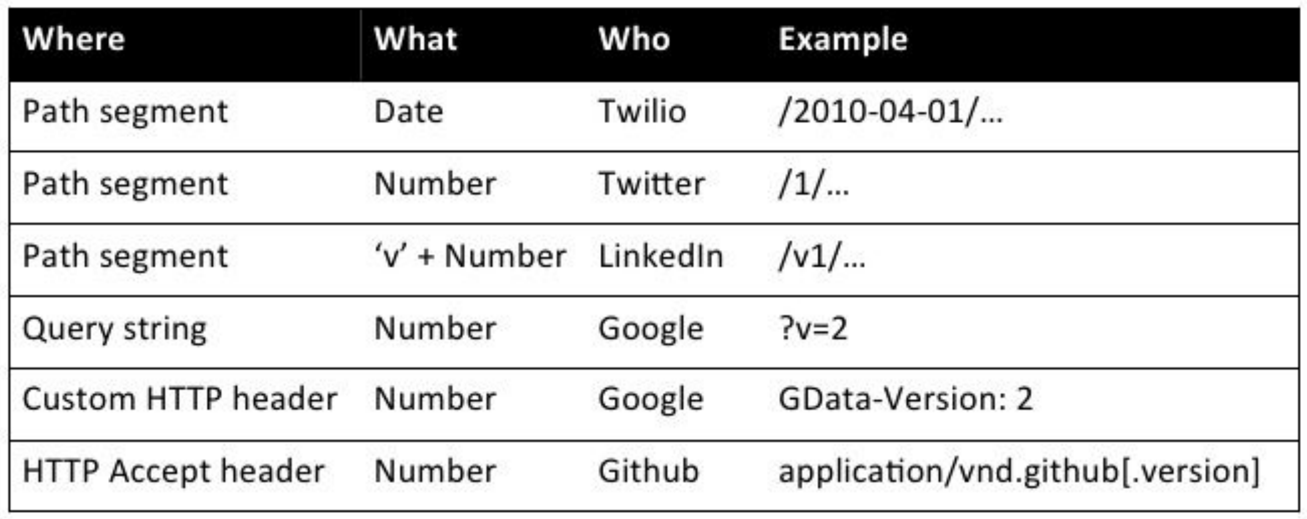
Versioning

- **Never release an API without a version**
  - Specify version with a “v” prefix (with the highest scope).
  - Use simple ordinal number (do not use dot notation).
- **Maintain at least one version back.**
  - Give developers at least one cycle to react before obsoleting a version (sometimes it is 6 months, sometimes it’s 2 years).



Versioning

- **Should version and format be in URLs or headers?**
    - If it changes the logic you write to handle the response put it in the URL (easier to notice)
    - Same resource, different logics



Where	What	Who	Example
Path segment	Date	Twilio	/2010-04-01/…
Path segment	Number	Twitter	/1/…
Path segment	'v' + Number	LinkedIn	/v1/…
Query string	Number	Google	?v=2
Custom HTTP header	Number	Google	GData-Version: 2
HTTP Accept header	Number	Github	application/vnd.github[.version]

```
dogs/1
Content-Type: application/json

dogs/1
Content-Type: application/xml

dogs/1
Content-Type: application/png
```

Pagination & Partial Response

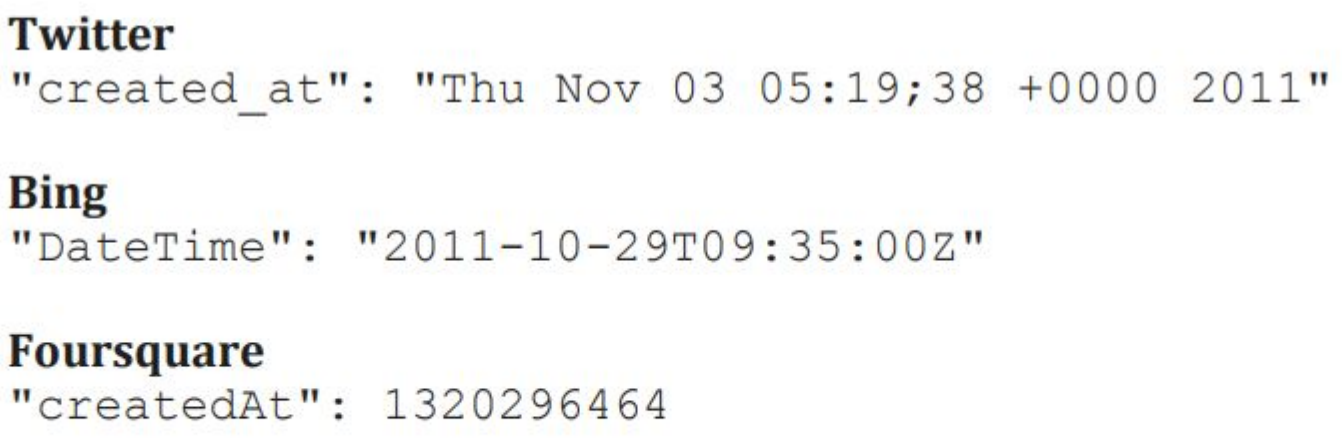
- **Partial response allows you to give developers just the information they need**
  - Facebook → /joe.smith/friends?fields=id,name,picture
  - Google → ?fields=title,media:group(media:thumbnail)
  - They each have an optional parameter called "fields" after which you put the names of fields you want to be returned.
    - Ex: `/dogs?fields=name,color,location`
- **Pagination**
  - It's almost always a bad idea to return every resource in a database.
  - Facebook uses offset and limit. Twitter uses page and rpp (records per page). LinkedIn uses start and count.
  - Use offset & limit (more common)
    - Ex: `/dogs?limit=25&offset=50`

Pagination & Partial Response

- **Metadata**
  - Ex: provide total number of records in each response.
- **Defaults**
  - You should define default values for offset & limit (Ex: limit=20, offset=0).
- **Summary**
  - Support partial response by adding optional fields in a comma delimited list.
  - Use limit and offset to make it easy for developers to paginate objects.

Attribute names & formats

- **Problem**



Twitter
```
"created_at": "Thu Nov 03 05:19;38 +0000 2011"
```

Bing
```
"DateTime": "2011-10-29T09:35:00Z"
```

Foursquare
```
"createdAt": 1320296464
```

- **Recommendations**
  - Use JSON as default
  - Follow JavaScript conventions for naming attributes
    - CamelCase
    - upper/lower case depending on type of object
    - Google JS Style Guide

Search

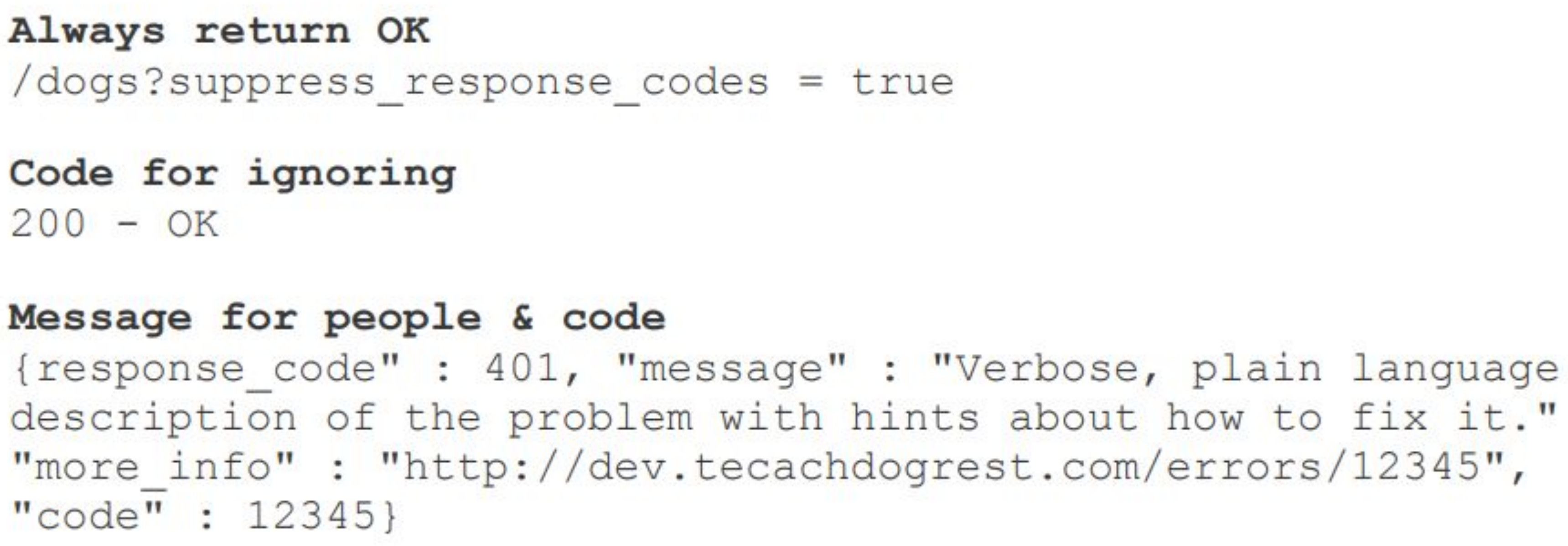
- **Global search**
  - If you want to do a global search across resources
  - Ex: `/search?q=fluffy+fur`
- **Scoped search**
  - To add scope to your search, you can prepend with the scope of the search.
  - For example, search in dogs owned by resource ID 56 `/owners/56/dogs?q=fluffy+fur`

Security

- **Every API endpoint needs to be authenticated**
  - HTTP (Basic, Bearer, etc.)
  - API keys (and cookie authentication)
  - OAuth 2
  - OpenID Connect Discovery
- **Protect your API from Attacks**
  - Denial of service → Rate limiting → HTTP 429
  - Ex: https://dev.bitly.com/docs/getting-started/rate-limits
  - others
- **DevSecOps**
  - Increase the level of security by prioritizing security at any stage or cycle of the DevOps pipeline.

Handling Exceptional Behavior

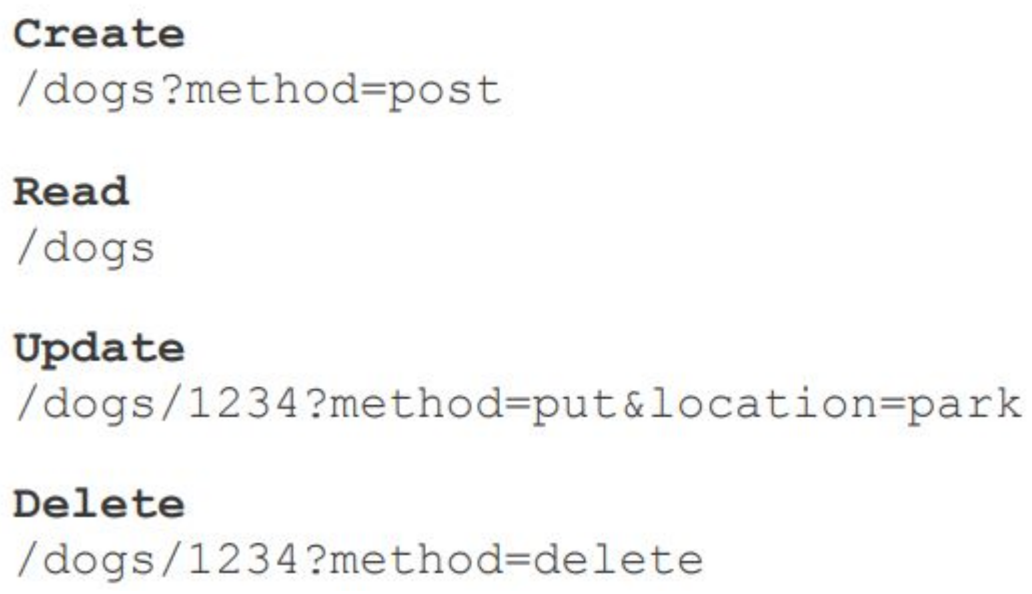
- **Client’s env. intercepts the response**
    - Use an optional parameter `suppress_response_codes`. If it is set to true, the HTTP response is always 200.



```
Always return OK
/dogs?suppress_response_codes = true

Code for ignoring
200 - OK

Message for people & code
{response_code" : 401, "message" : "Verbose, plain language
description of the problem with hints about how to fix it."
"more_info" : "http://dev.tecachdogrest.com/errors/12345",
"code" : 12345}
```

Handling Exceptional Behavior

- **Client supports limited HTTP methods**
  - Ex: support for GET and POST and not PUT and DELETE.
  - To maintain the integrity of the four HTTP methods, parameters can be used to set the logic (Warning: it can be dangerous).



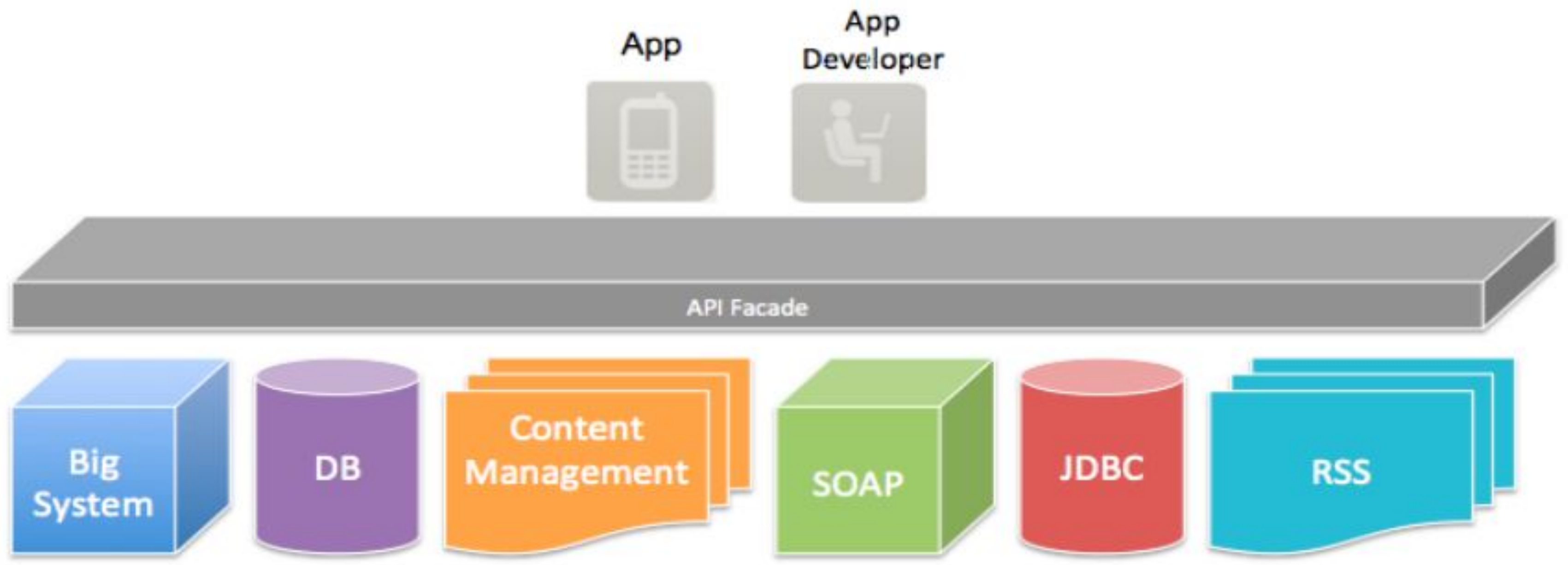
```
Create
/dogs?method=post

Read
/dogs

Update
/dogs/1234?method=put&location=park

Delete
/dogs/1234?method=delete
```

Other Recommendations

- **Consolidate reqs. under one API subdomain**
  - It's cleaner, easier and more intuitive for developers who you want to build cool apps using your API (Ex: api.techdogs.com)
- **Do web redirects**
  - Say a developer types `api.teachdogs.com` in the browser but there's no other information for the GET request, you can probably safely redirect to your developer portal.
- **Use JSON Objects for all**
  - In a response body, you must always return a JSON object (and not e.g. an array) as a top level data structure to support future extensibility.
    - ~~`[ "Ford", "BMW", "Fiat" ]`~~
    - `{"cars":[ "Ford", "BMW", "Fiat" ]}`

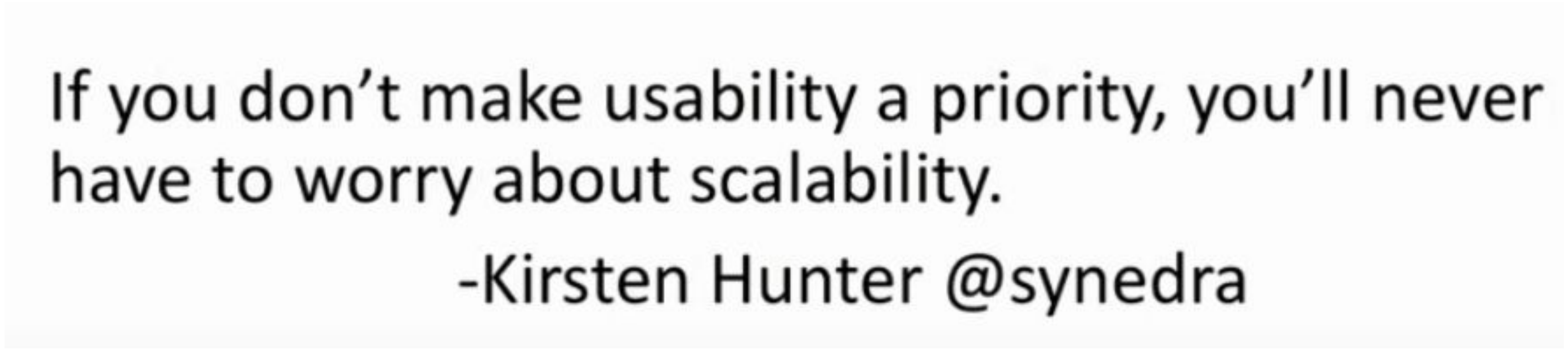
Designing the interface

- **API Façade Pattern**
  - A simple interface to complex system
  - Façades hide the complexity of internal implementations and provide simple interfaces.



References

- Zalando RESTful API and Event Scheme Guidelines: [link]
- Microsoft REST API Guidelines: [link]
- https://httpstatuses.com/
- APIGee Web API Design: [link]
- APIGee API Façade Pattern: [link]


If you don't make usability a priority, you'll never have to worry about scalability.

-Kirsten Hunter @synedra

https://www.slideshare.net/SpencerSchneidenbach/restful-api-design-best-practices-using-aspnet-web-api

Q/A