CEU | Universidad San Pablo

REST

Sistemas Web II

Grado en Ingeniería de Sistemas de Información

Álvaro Sánchez Picot
alvaro.sanchezpicot@ceu.es

v20230323

Basado en el trabajo de:

- David González Márquez

Índice

- REST
- YAML
- OpenAPI
- Node.js
- Otras Alternativas

Why talk about REST?

Because

REST

has become a

BUZZWORD

There's nothing particularly wrong with that...
unless you happen to be me...
or working with me

Absence of occupation is not REST,
A mind quite vacant is a mind distress'd. [William Cowper]

Day

The Web Problem (circa 1994)

Early architecture based on solid principles

- URLs, separation of concerns, simplicity
  - lacked architectural description and rationale

Protocols assumed a direct server connection

- no awareness of caching, proxies, or spiders
- many independent extensions

Emerging awareness of the Web

- exponential growth threatened the Internet
  - commercialization meant new stakeholders with new (selfish) requirements

A modern Web architecture was needed

- but how do we avoid breaking the Web in the process?

REST is not quitting the busy career;
REST is the fitting of self to its sphere. [John Sullivan Dwight]

Day

Web Requirements & Properties

Low entry barrier

- Hypermedia User Interface
- Simple protocols for authoring and data transfer

▸ must be **Simple** and **Reusable**; want **Extensible**

Distributed Hypermedia System

- Large data transfers
- Sensitive to user-perceived latency

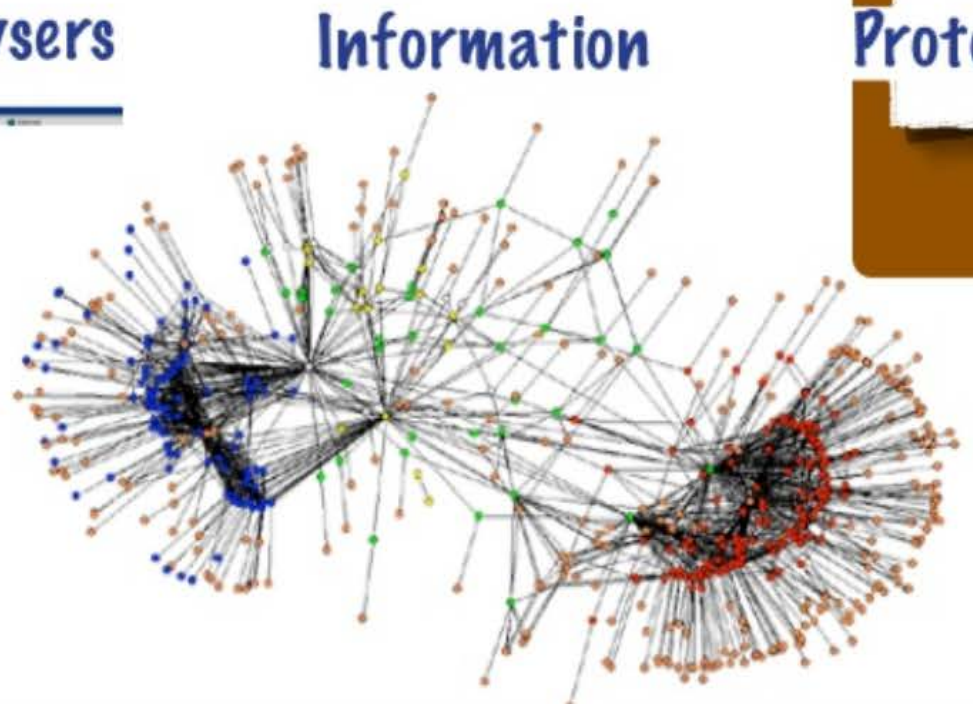
▸ must be **Data-driven** and **Streamable**; want **Performant**

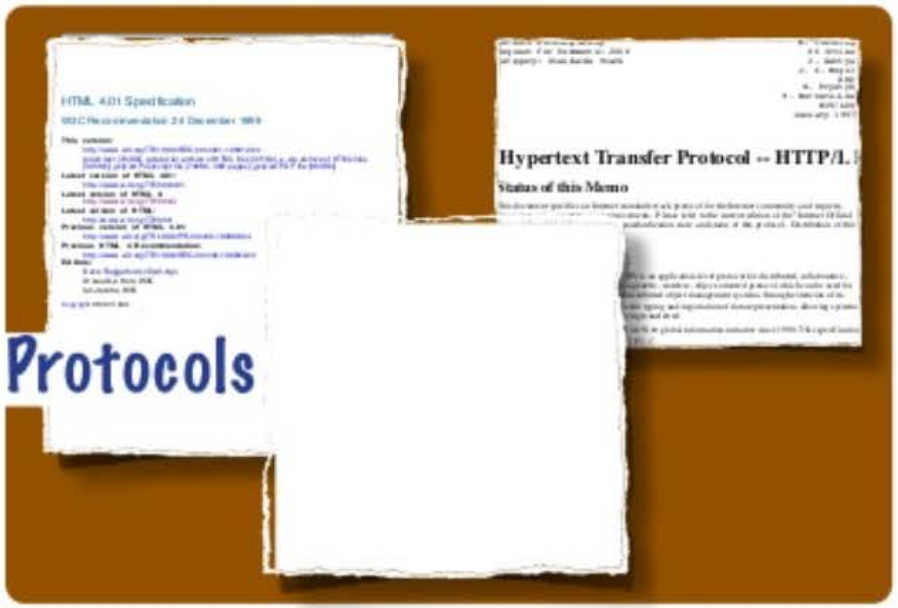
Multiple organizational boundaries

- Anarchic scalability
- Gradual and fragmented change (deployment)

▸ must be **Scalable**, **Portable**, **Evolvable**; want **Reliable**, **Visible**, **Customizable**, **Configurable**, **Extensible**, ...

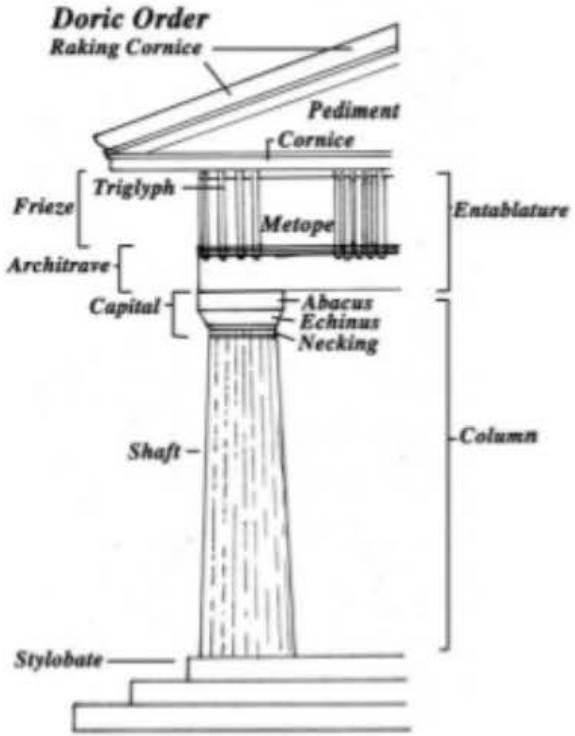
Three (very different) perspectives of the Web

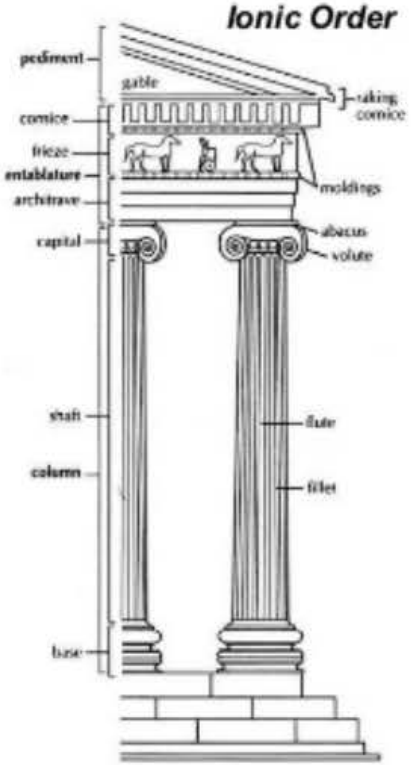




Architectural Styles

- A **horizontal abstraction** across multiple architectures (vertical abstractions)
  - names a repeated architectural pattern
  - defined by its design constraints
  - chosen for the properties they induce





- REST is an architectural style
  - For network-based applications
  - To induce the same architectural properties as the World Wide Web

REST

is NOT an

implementation

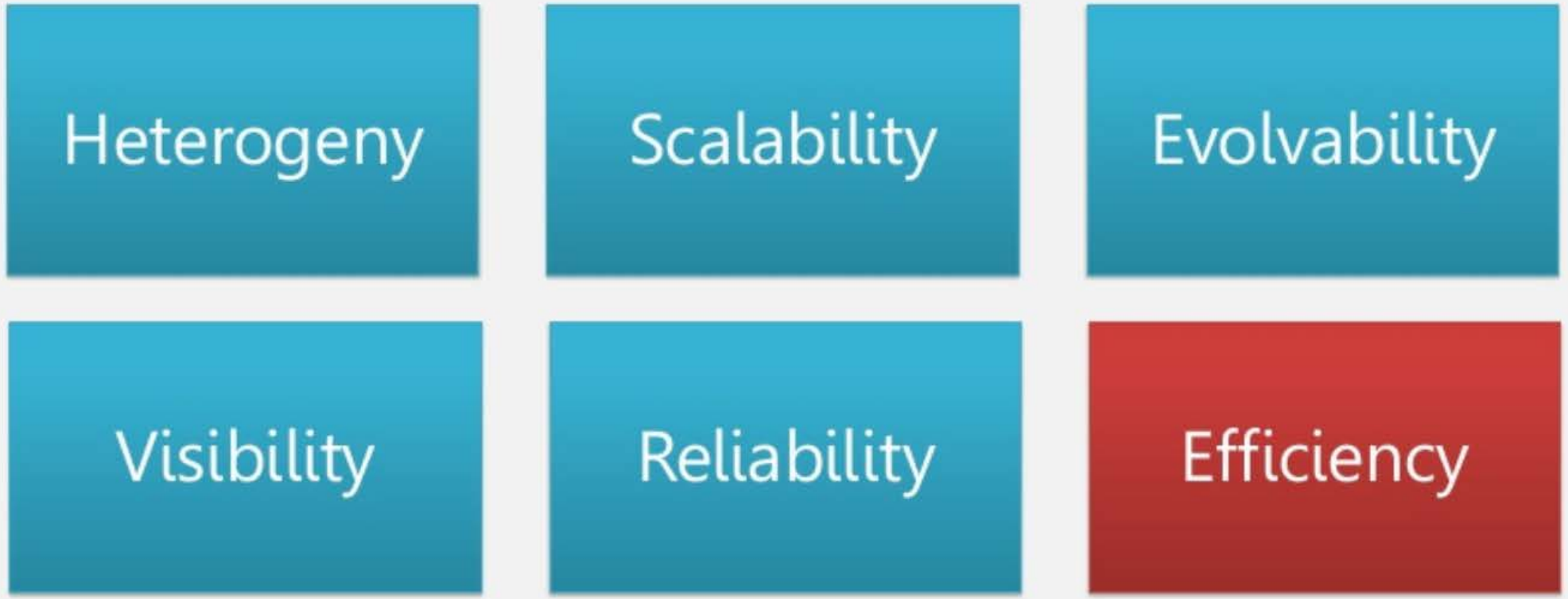
Arquitectura REST style

- REST: REpresentational State Transfer
- Definido por Roy Fielding en su tesis doctoral en el año 2000
  - Desarrollado en paralelo a HTTP 1.1
- Independiente del protocolo
  - Soporte de URIs
  - Principalmente usado con HTTP
- Estilo de arquitectura
  - No es un estándar (no hay RFC)
  - No hay "hard rules"

Arquitectura REST style

- Características principales:
  - Sigue el paradigma cliente-servidor
  - Sin estado
  - Sistema basado en capas
  - Interfaz uniforme y desacoplada.
  - Hiperenlace como motor
  - Cacheable
- Orientado al **recurso**

Properties of REST



Arquitectura REST style

Un estilo de arquitectura es una serie de restricciones que limitan y moldean la forma de la arquitectura y las relaciones entre los elementos en cualquier arquitectura que sigue el estilo

Arquitectura REST style

Perspectives on the process of architectural design:

- Start with nothing
  - Build-up an architecture from familiar components
  - Until it satisfies the needs of the intended system
  - Emphasizes creativity and unbounded vision
- Start with the system needs as a whole
  - Start without constraints
  - Incrementally identify and apply constraints
  - Emphasizes restraint and understanding of the system context

How beautiful it is to do nothing,
and then REST afterward. [Spanish Proverb]

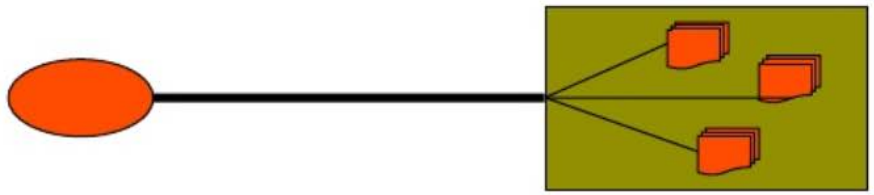
Day

Style = nil

Starting from a condition of no constraints...

Style += Client/Server

Apply separation of concerns: Client-Server



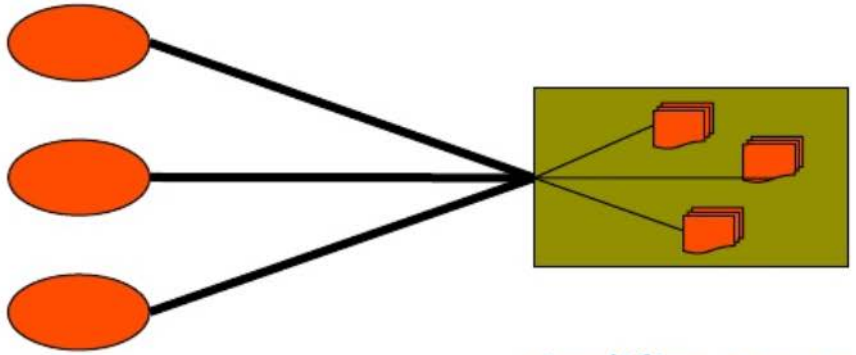
improves UI portability

simplifies server

enables multiple organizational domains

Style += Stateless

Constrain interaction to be stateless...



simplifies server

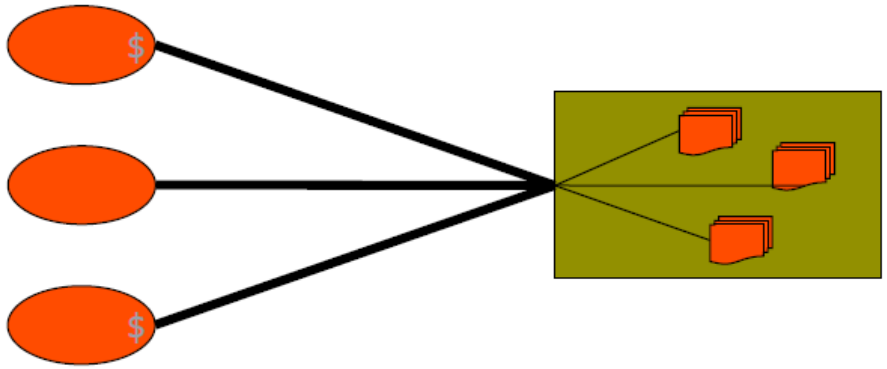
improves scalability

improves reliability

degrades efficiency

Style += Caching

Add optional non-shared caching



reduces average latency

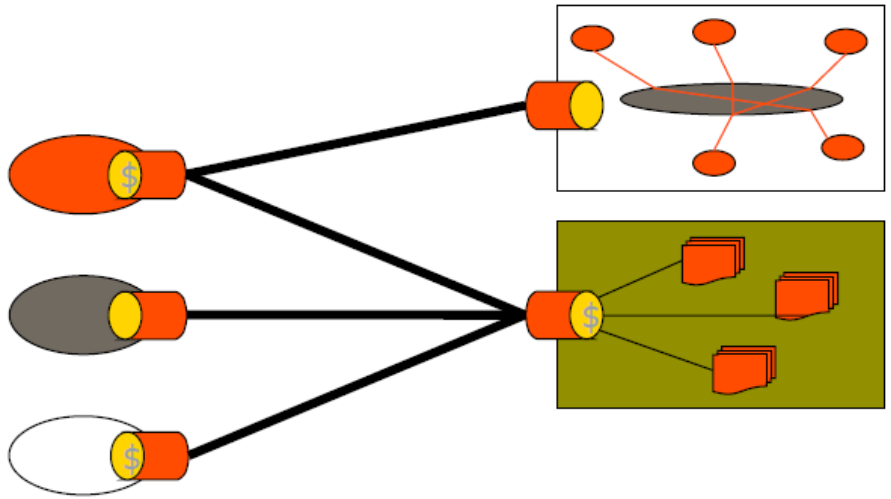
degrades reliability

improves efficiency

improves scalability

Style += Uniform Interface

Apply generality: uniform interface constraint



improves visibility

degrades efficiency

independent evolvability

decouples implementation

REST Uniform Interface

All important resources are identified by one (uniform) resource identifier mechanism

- simple, visible, reusable, stateless communication

Access methods (actions) mean the same for all resources (universal semantics)

- layered system, cacheable, and shared caches

Resources are manipulated through the exchange of representations

- simple, visible, reusable, cacheable, and stateless communication

Exchanged as self-descriptive messages

- layered system, cacheable, and shared caches

REST Uniform Interface

Hypertext as the engine of application state

- A successful response indicates (or contains) a current **representation** of the state of the identified resource; the **resource remains hidden** behind the interface.
- Some **representations contain links** to potential next application states, including **direction on how to transition** to those states when a transition is selected.
- Each steady-state (Web page) embodies the current **application state**
  - simple, visible, scalable, reliable, reusable, and cacheable
- All application state (not resource state) is kept on client
- All shared state (not session state) is kept on origin server

Hypertext Clarification

Hypertext has many (old) definitions

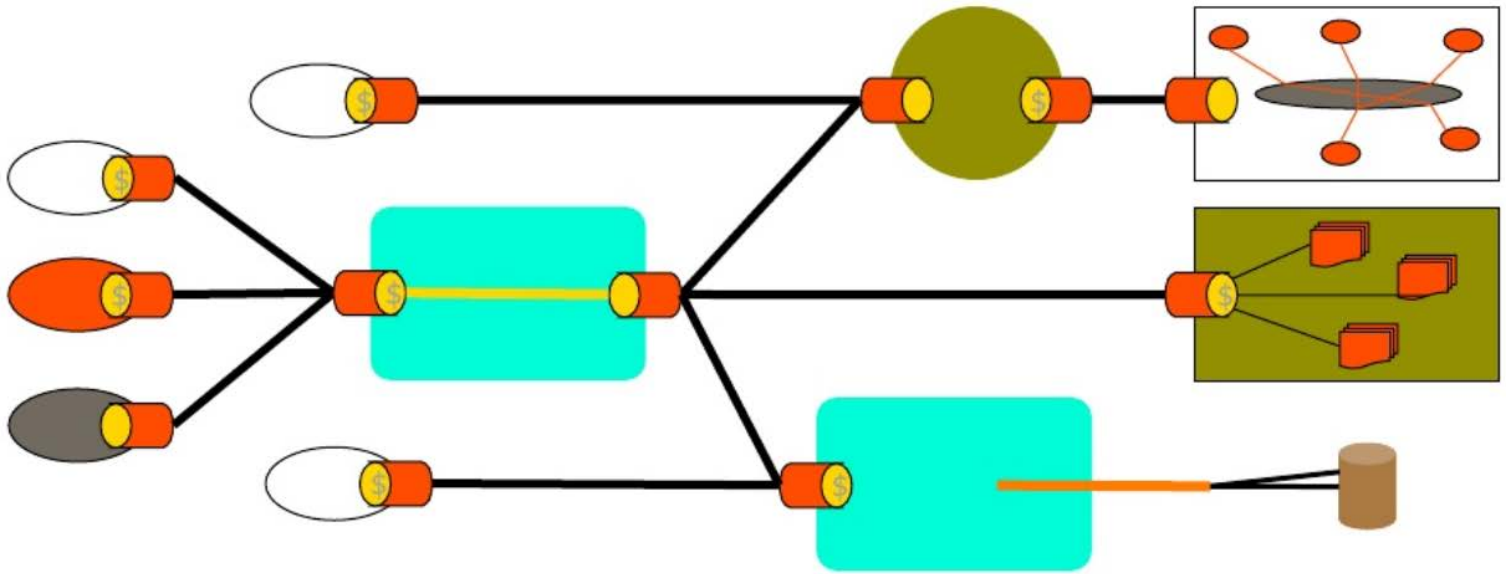
- "By 'hypertext,' I mean non-sequential writing — text that branches and allows choices to the reader, best read at an interactive screen. As popularly conceived, this is a series of text chunks connected by links which offer the reader different pathways"
  [Theodor H. Nelson]
- "Hypertext is a computer-supported medium for information in which many interlinked documents are displayed with their links on a high-resolution computer screen."
  [Jeffrey Conklin]

When I say Hypertext, I mean ...

- The simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions.
- Hypertext does not need to be HTML on a browser
  - machines can follow links when they understand the data format and relationship types

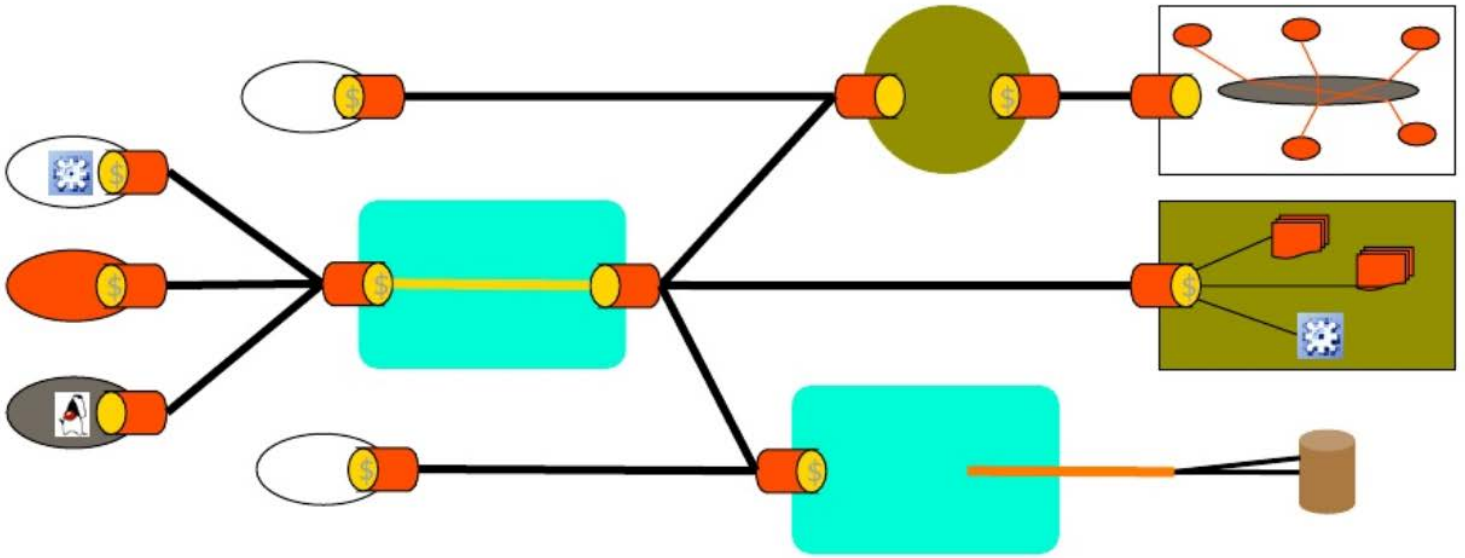
Style += Layered System

Apply info hiding: layered system constraints



REST Style

Finally, allow code-on-demand (applets/js)



simplifies clients

improves extensibility

reduces visibility

Constraints

Interfaz Uniforme

- La definición de los recursos consta de:
  - **URI**
  - **Formato de intercambio de datos** (JSON, XML)
  - **Métodos** (GET,PUT,POST,…)
- Los recursos se identifican de forma unívoca por su URI
- Los servicios permiten leer, crear, modificar y eliminar recursos.
- Hypermedia As The Engine Of Application State (HATEOAS)

Interfaz Uniforme

Las operaciones se corresponden con los métodos HTTP

- **GET:** obtener un recurso
- **POST:** crear un recurso nuevo
- **PUT:** crear/actualizar un recurso
- **DELETE:** eliminar un recurso

Interfaz Uniforme

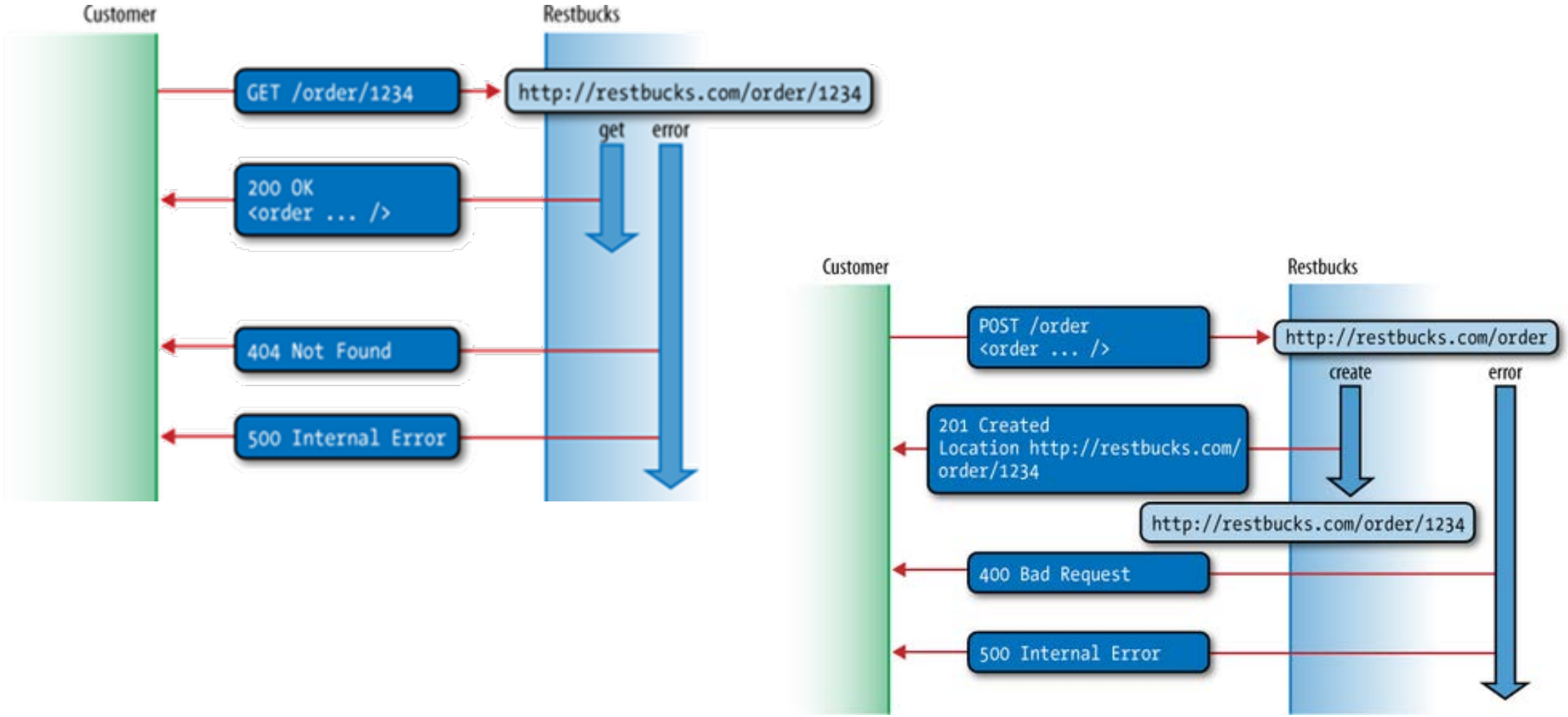
Verb	Path	Action	Used for
GET	/photos	index	display a list of all photos' information
GET	/photos/:id	show	display specific photo
GET	/photos/new	new	return an HTML form for creating a new photo
GET	/photos/:id/edit	edit	return an HTML form for editing a photo
POST	/photos	create	create a new photo
PUT	/photos/:id	update	update a specific photo
DELETE	/photos/:id	destroy	delete a specific photo

Interfaz Uniforme

Los códigos de estado corresponden a los de HTTP:

- 200 OK: Petición finalizó con éxito
- 201 Created: La petición POST finalizó correctamente y el objeto fue creado, se debe acompañar con la URI del nuevo objeto
- 204 No content: La petición finalizó con éxito pero no devuelve ningún cuerpo de mensaje
- 400 Bad request: La petición no se pudo realizar por estar mal formateada
- 401 Unauthorizaed: Faltan datos de autenticación y/o autorización
- 403 Forbidden: Autenticación OK, pero no se tiene acceso al recurso solicitado
- 404 Not found: El recurso solicitado no existe
- 405 Method not allowed: El método HTTP no está soportado en la URI dada

Interfaz Uniforme



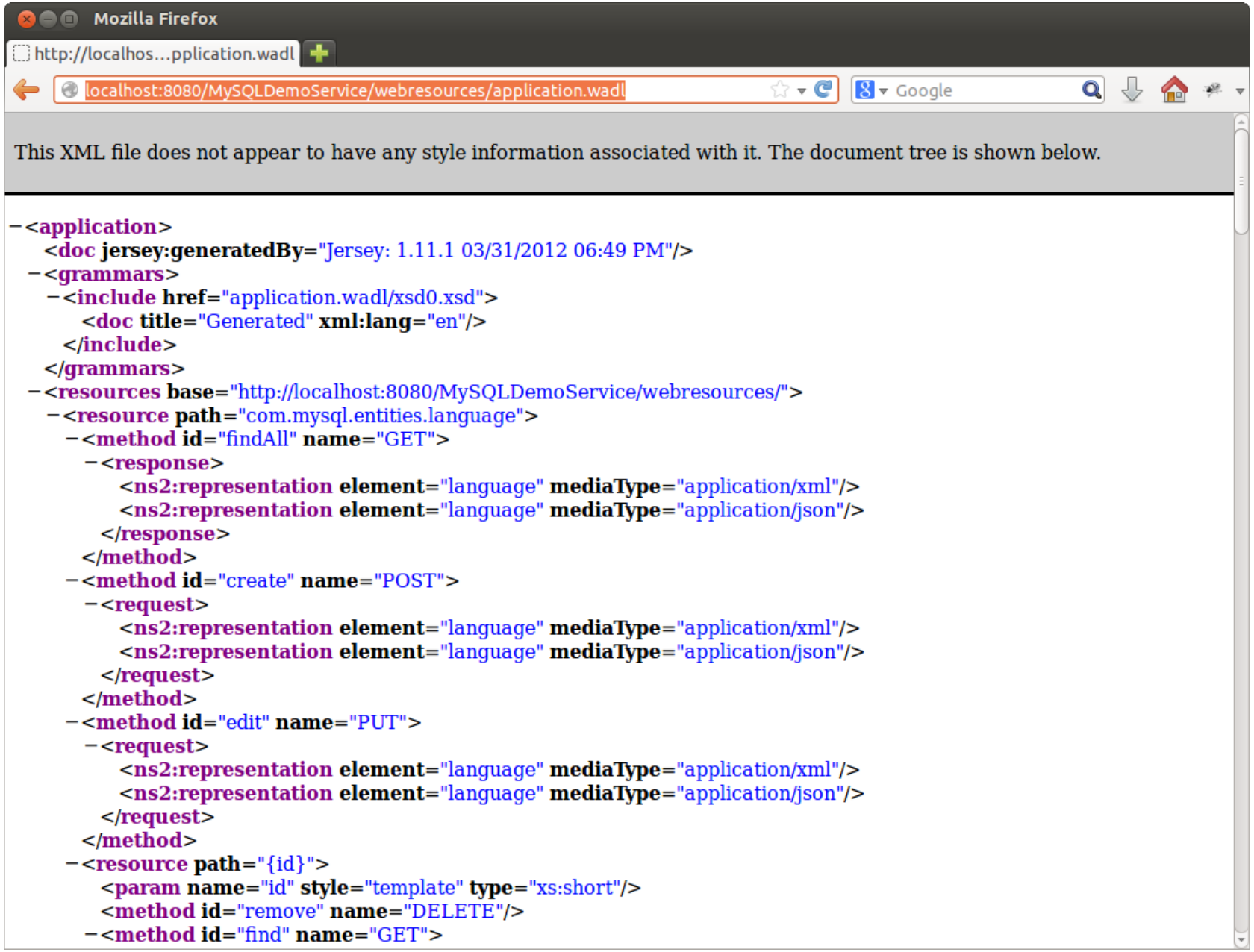
Orientado al recurso

- Con HTTP el recurso se representa en el cuerpo del mensaje
- El tipo de formato se especifica mediante el tipo de contenido
- El cliente puede negociar con Accept
- Content-Type
  - text/plain
  - text/html
  - application/xml
  - application/json

Orientado al recurso

Contrato de servicio WADL:

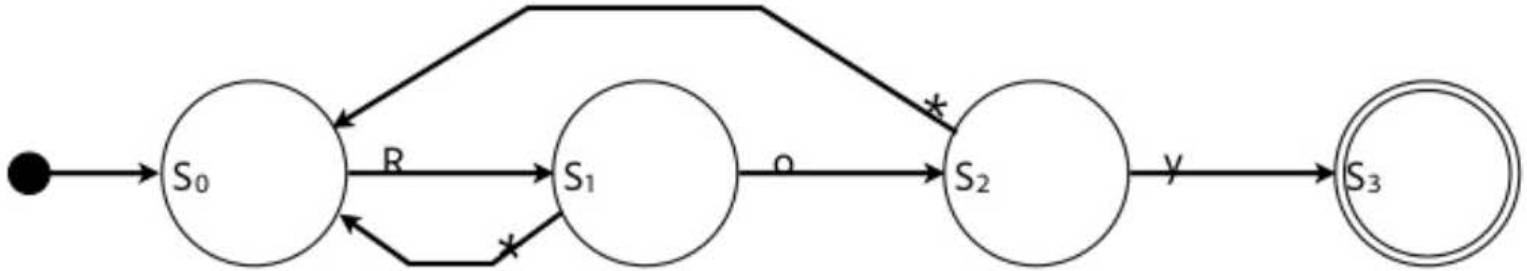
- Equivalente a WSDL de SOAP
- Su uso es opcional, no demasiado extendido, se puede usar WSDL también
- Útil para generar código de cliente y de servidor
- Mejor integración con herramientas y frameworks
- Hay otros estándares mejores como OpenAPI (Swagger)


Mozilla Firefox
http://localhos...pplication.wadl
localhost:8080/MySQLDemoService/webresources/application.wadl
Google
This XML file does not appear to have any style information associated with it. The document tree is shown below.
−<application>
<doc jersey:generatedBy="Jersey: 1.11.1 03/31/2012 06:49 PM"/>
−<grammars>
−<include href="application.wadl/xsd0.xsd">
<doc title="Generated" xml:lang="en"/>
</include>
</grammars>
−<resources base="http://localhost:8080/MySQLDemoService/webresources/">
−<resource path="com.mysql.entities.language">
−<method id="findAll" name="GET">
−<response>
<ns2:representation element="language" mediaType="application/xml"/>
<ns2:representation element="language" mediaType="application/json"/>
</response>
</method>
−<method id="create" name="POST">
−<request>
<ns2:representation element="language" mediaType="application/xml"/>
<ns2:representation element="language" mediaType="application/json"/>
</request>
</method>
−<method id="edit" name="PUT">
−<request>
<ns2:representation element="language" mediaType="application/xml"/>
<ns2:representation element="language" mediaType="application/json"/>
</request>
</method>
−<resource path="{id}">
<param name="id" style="template" type="xs:short"/>
<method id="remove" name="DELETE"/>
−<method id="find" name="GET">

Sin Estado

- El servidor no debe mantener el estado del cliente
  - Simplifica el balanceo de carga
  - Permite el uso de cachés
- El servidor almacena únicamente el estado de sus recursos
- La sesiones del lado del servidor no están permitidas
  - El cliente sí puede gestionar él una sesión
  - Cualquier petición realizada desde el cliente tendrá que contener toda la información que se necesite para poderla procesar en el servidor

Hypertext as the Engine of Application State

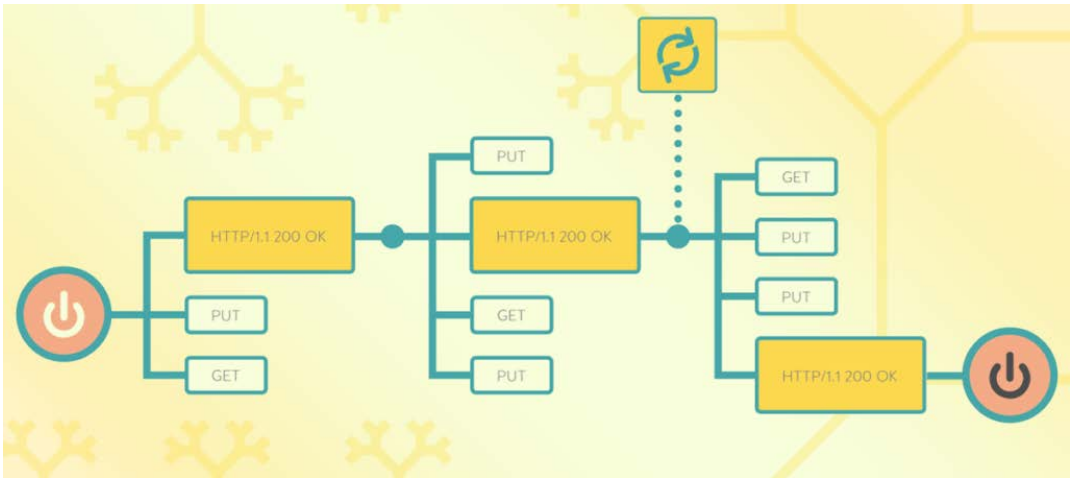


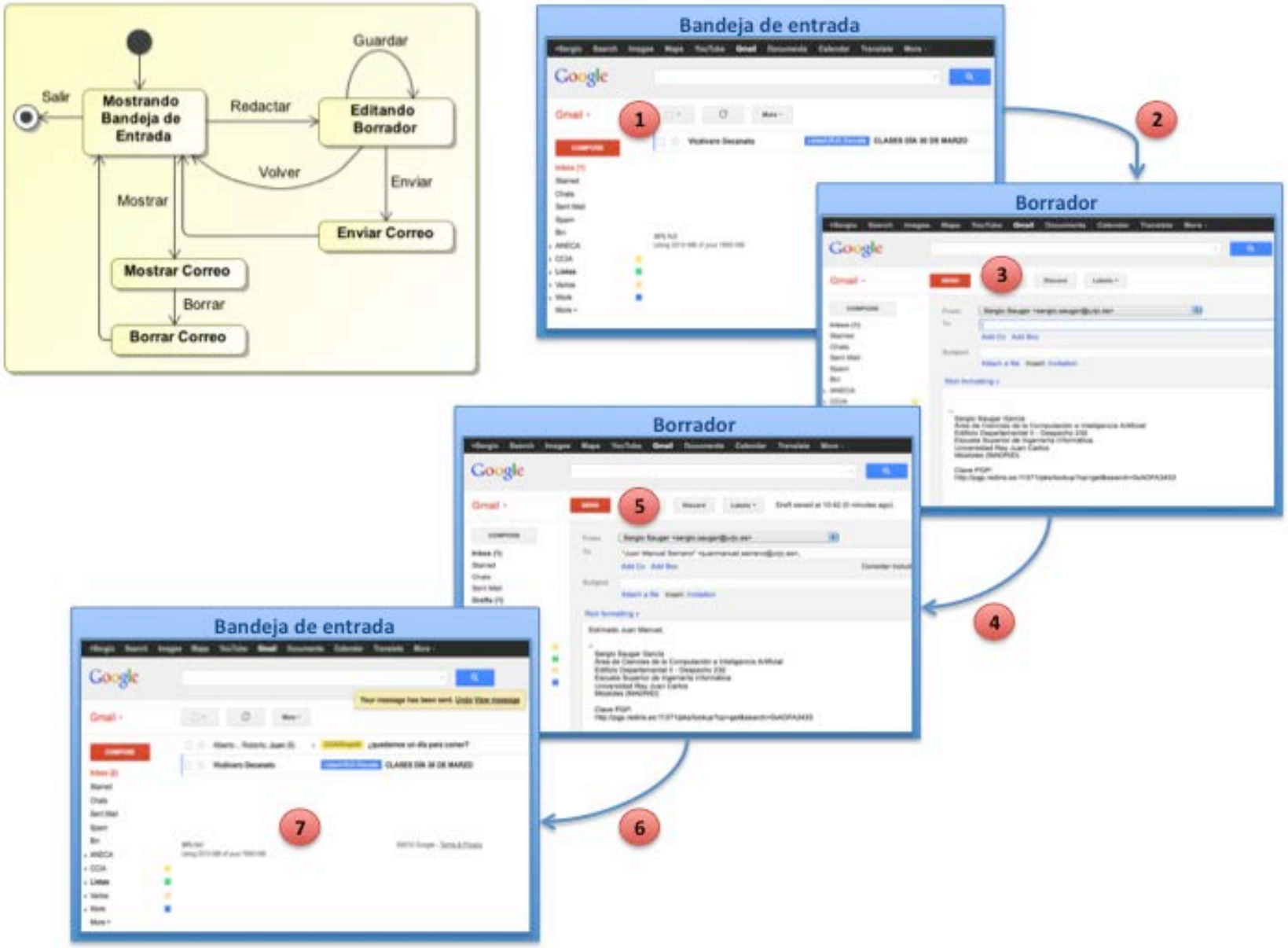
each state can be dynamic

each transition can be redirected

HATEOAS

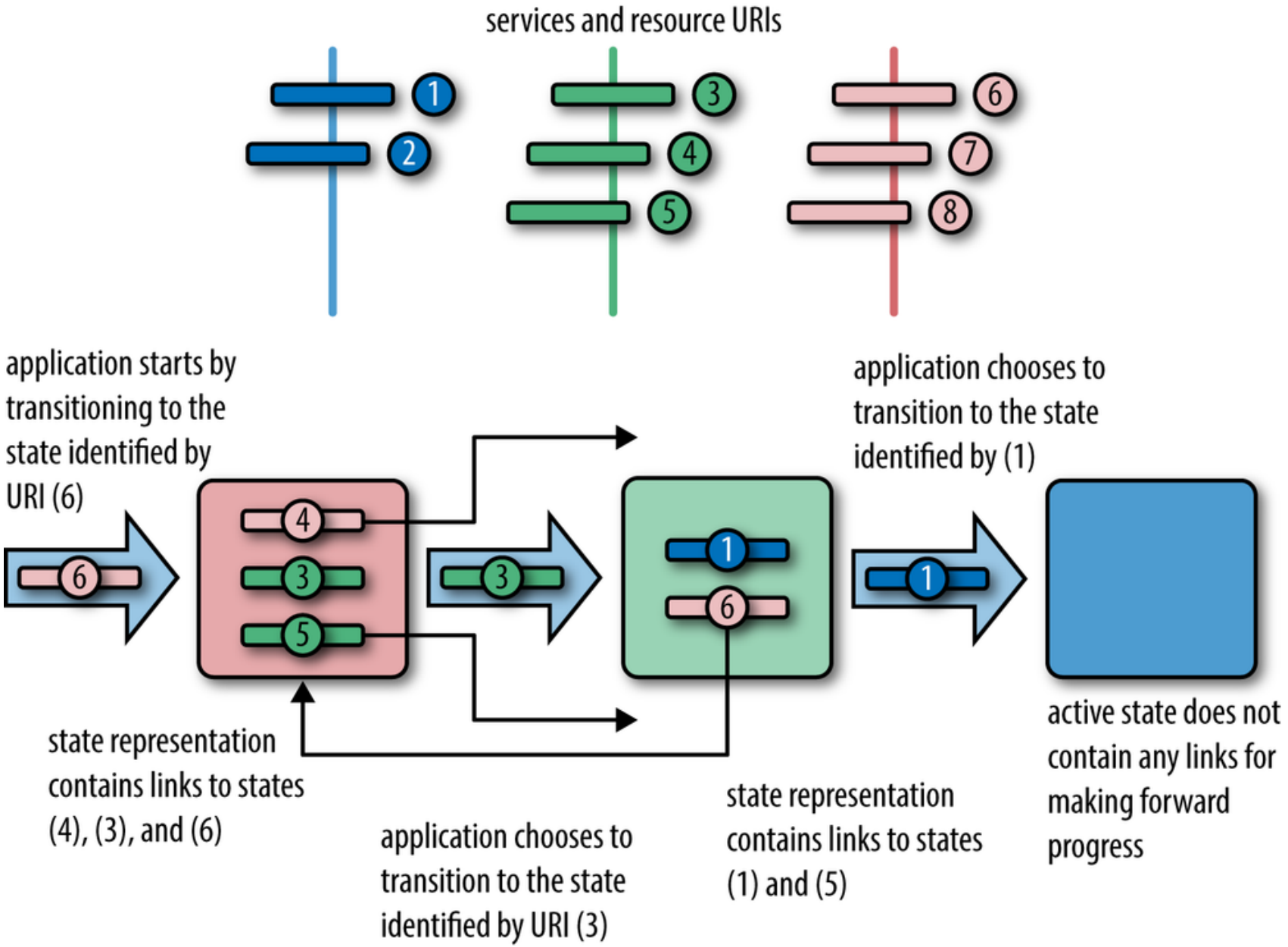
- En REST una aplicación web se puede ver como una gran máquina virtual de estados
  - Los distintos recursos son los nodos de la máquina
  - Los enlaces representan las transiciones de estado
  - El siguiente estado de la máquina será el resultado de seguir el enlace y el servicio al que se acceda




Guardar
Salir
Mostrando Bandeja de Entrada
Redactar
Editando Borrador
Volver
Enviar
Mostrar
Enviar Correo
Mostrar Correo
Borrar
Borrar Correo
Bandeja de entrada
1
2
Borrador
3
Borrador
5
4
Bandeja de entrada
7
6

HATEOAS

- Hypermedia As The Engine Of Application State
- Extensión del concepto de hipervínculo
- Los recursos tienen enlaces que apuntan a su vez a otros recursos
- La aplicación sabe tras cada respuesta cuáles son las posibles continuaciones
- El cliente mantiene el estado de la aplicación (máquina de estados implícita)


services and resource URIs
1
2
3
4
5
6
7
8
application starts by transitioning to the state identified by URI (6)
application chooses to transition to the state identified by (1)
6
4
3
5
3
1
6
1
state representation contains links to states (4), (3), and (6)
application chooses to transition to the state identified by URI (3)
state representation contains links to states (1) and (5)
active state does not contain any links for making forward progress

Seguridad en REST

- La seguridad en servicios REST emplea los mecanismos disponibles en las capas de transporte (SSL/TLS) y HTTP
- Mecanismos de autenticación HTTP
  - HTTP Basic
  - HTTP Digest (not widely supported)
  - X.509 certificates via SSL
- SSL permite emplear cifrado y firma
- Limitación frente a las soluciones de seguridad basada en mensaje proporcionadas por WS-Security

Benefits of REST-based Architecture

Maximizes reuse

- uniform resources having identifiers = Bigger WWW
- visibility results in serendipity

Minimizes coupling to enable evolution

- uniform interface hides all implementation details
- hypertext allows late-binding of application control-flow
- gradual and fragmented change across organizations

Eliminates partial failure conditions

- server failure does not befuddle client state
- shared state is recoverable as a resource

Scales without bound

- services can be layered, clustered, and cached

Benefits of REST-based Architecture

Simplifies

- hypertext is standardized (fewer UIs)

Simplifies

- identification is standardized (less communication)

Simplifies

- exchange protocols are standardized (fewer integrations)

Simplifies

- interactions are standardized (fewer semantics)

Simplifies

- data formats are standardized (fewer translations)

Ventajas de REST

- Utiliza tecnologías y plataformas bien documentadas y maduras (HTTP)
- Facilidad de uso, dada una URI todo el mundo sabe como acceder a ella
- Nos evitamos añadir otro protocolo
- Permite utilizar una gran cantidad de formatos en el mensaje
- Hereda la seguridad de HTTP, tecnología madura
- “Makes sense” Usa lo que ya está establecido para dar un paso más, es la extensión lógica de la web

Industry Practice

Meanwhile, in a parallel universe ...

- Monty Python's Architect Sketch
  - Microsoft was selling COM+/DCOM
  - IBM and friends were selling CORBA
  - Sun was selling RMI
  - W3C was developing XML
- Then SOAP was dropped on the shower floor as an Internet Draft
  - and quickly laughed out of the IETF
  - only to be picked up by IBM and renamed "Web Services"
- and REST became the only counter-argument to multi-billions in advertising

Industry Reaction?

Not very constructive

- proponents labeled as RESTafarians
- arguments derided as a “religion”
- excused as “too simple for real services”

Service-Oriented Architecture (SOA)

- a direct response to REST
- attempt at an architectural style for WS
  - without any constraints
- What is SOA?
  - Wardrobe, Musical Notes, or Legos?
  - http://www.youtube.com/profile_videos?user=richneckyogi

Industry Acceptance

Something has changed ...

- People started to talk about the value of URIs (reusable resources)
- Google maps decided to encourage reuse (Mashups)
- O'Reilly began talking about Web 2.0
- Rails reminded people that frameworks can be simple

and REST(ful) became an industry buzzword

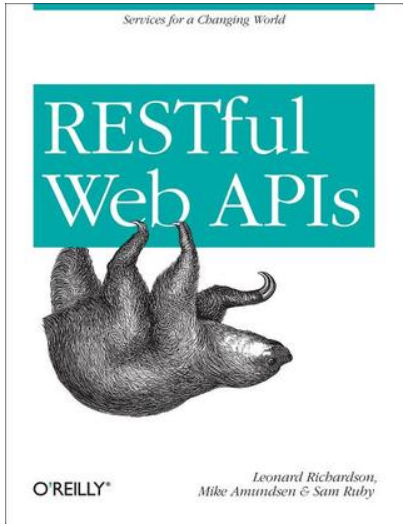
Yikes!

Servicios RESTful

Leonard Richardson

"As with any other technology, the Web will **not automatically solve** a business's application and integration problems"

"But **good design practices and adoption of good, well-tested, and widely deployed patterns** will take us a long way in our journey to build great web services"

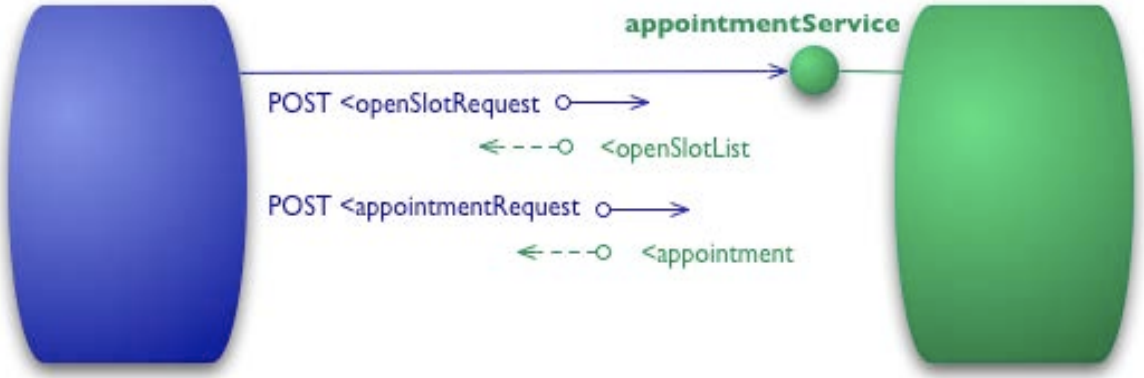


Servicios RESTful

- Richardson propuso una clasificación de servicios de la Web, lo que se conoce como un modelo de madurez “maturity model”
- En esta clasificación tenemos varios niveles, basados en el soporte de un servicio para URIs, HTTP y hipermedia.
- Es una clasificación general que nos permite describir todos los servicios web

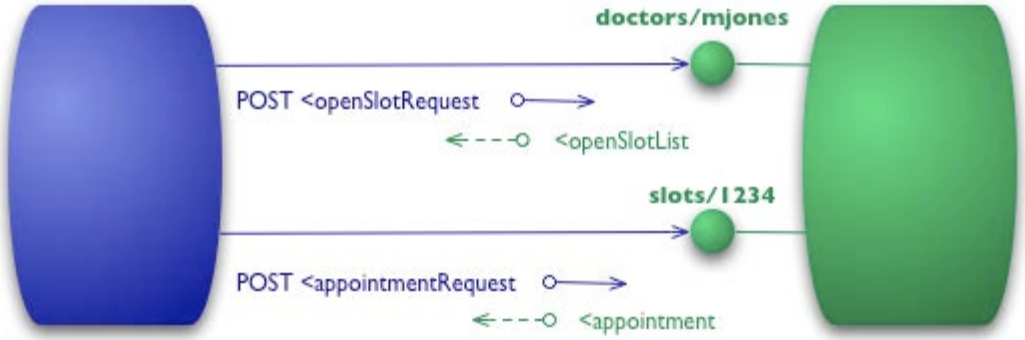
Servicios RESTful – Nivel 0

- El nivel más básico
- Servicios que tienen una única URI
- Que solo utilizan un único método HTTP (POST típicamente)
- Ej: Servicios SOAP
- Se usa HTTP pero sin aprovechar las ventajas y las opciones que da (“tunelling” using HTTP)



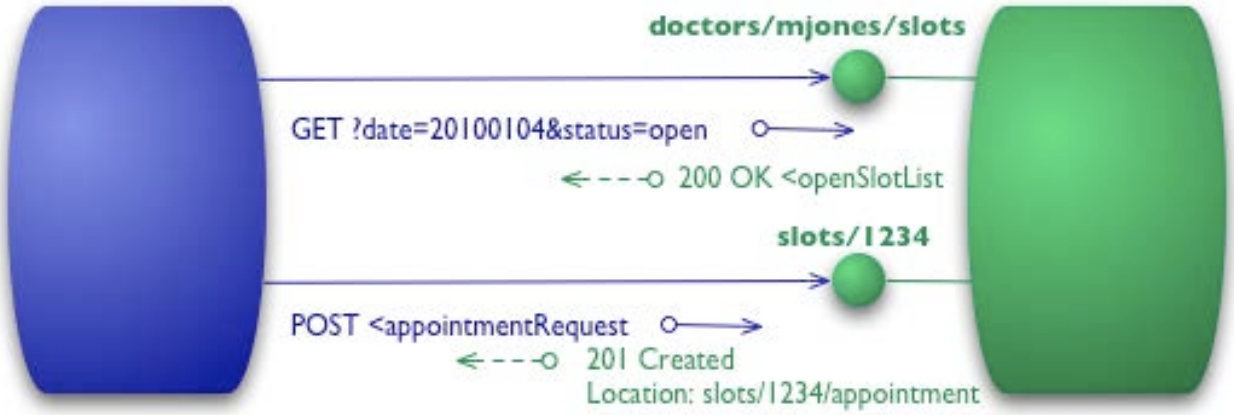
Servicios RESTful – Nivel 1

- Utilizamos varias URIs pero únicamente un método HTTP (normalmente GET o POST)
- Se exponen varios recursos
- Muchos de los servicios REST actuales se quedan en este nivel



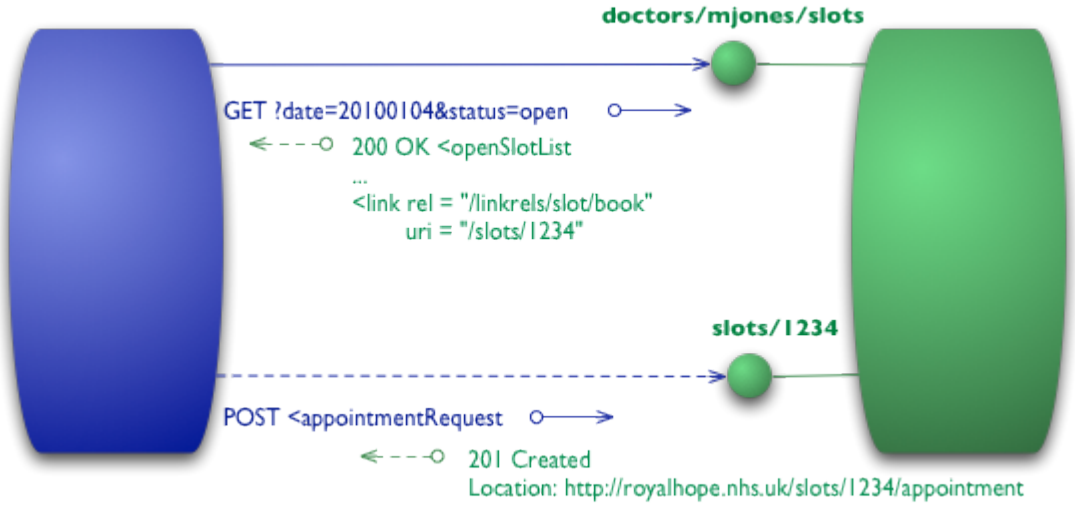
Servicios RESTful – Nivel 2

- Utilizamos varias URIs que además soportan varios métodos HTTP
- Tenemos servicios que soportan las operaciones CRUD
- También se utilizan los códigos de estado HTTP
- Dejamos de utilizar HTTP simplemente como transporte

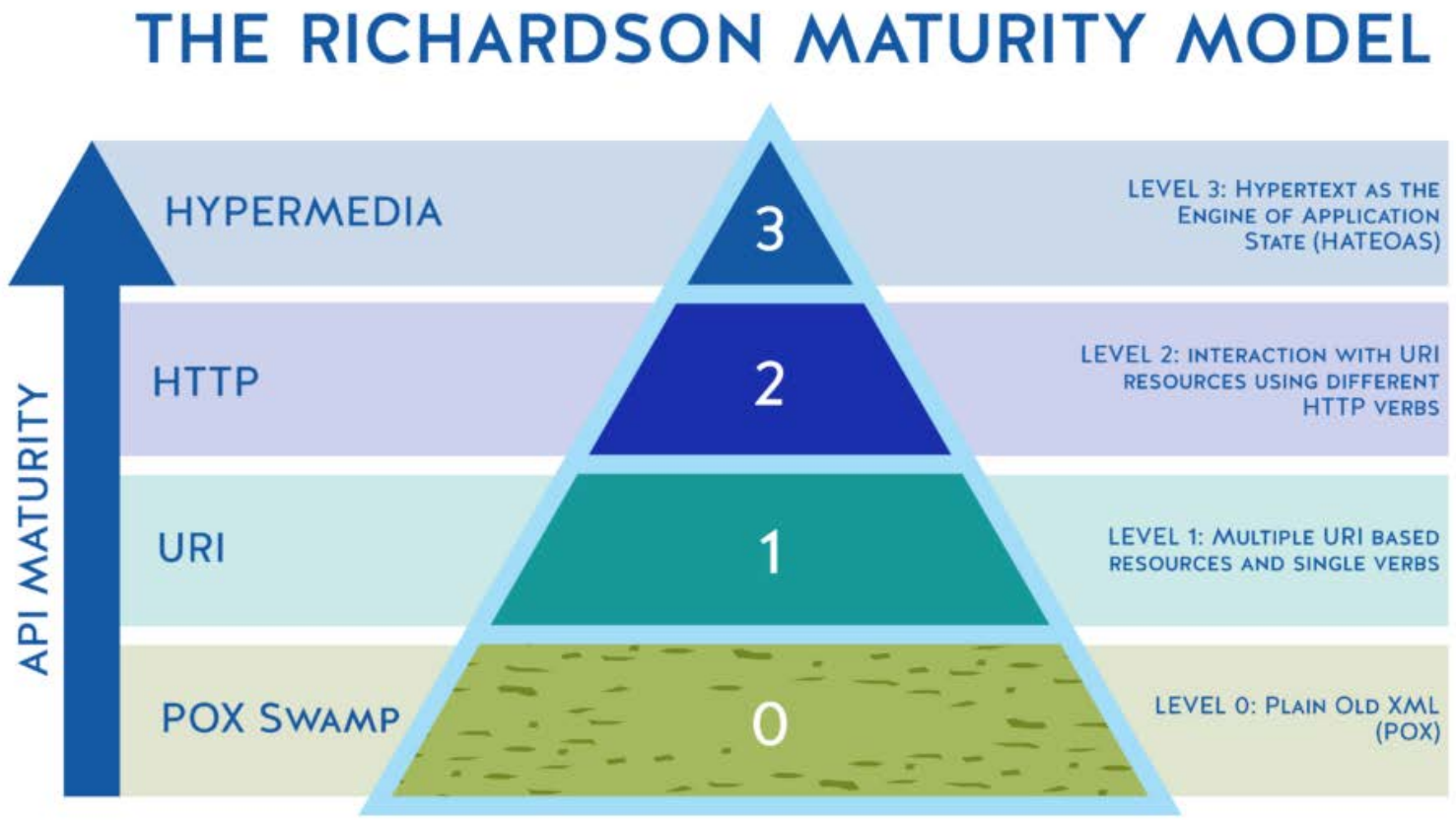


Servicios RESTful – Nivel 3

- Se añade la noción de HATEOAS
- Las representaciones incluyen URIs que hacen referencia a otros recursos, la transición de un recurso a otro genera el estado de la aplicación



Servicios RESTful



Buenas prácticas en REST

- Nombrado: sustantivos, no verbos

  `http://example.com/api/~~getBooks~~` → `http://example.com/api/books`

- Múltiples URIs por recurso

  `http://example.com/api/books o http://example.com/api/books/10`

  `GET http://example.com/api/books` → Obtener todos los libros

  `GET http://example.com/api/books/10` → Obtener el libro con ID 10

  `POST http://example.com/api/books` → Crear un nuevo libro

  `PUT http://example.com/api/books/10` → Actualizar libro con ID 10

  `DELETE http://example.com/api/books` → Borrar todos los libros

  `DELETE http://example.com/api/books/10` → Borrar libro con ID 10

Buenas prácticas en REST

- Asociaciones. La API debe ser intuitiva a la hora de definir las asociaciones

  `http://example.com/api/user/23/books/10` → Obtener el libro con ID 10 para el usuario con ID 23
- No sobrecargar al cliente, si el contenido que se va a servir es muy grande usar paginación (no devolver todos los resultados)
- Recordar que las peticiones GET tienen un tamaño máximo
- Si el servicio tiene muchos argumentos o con valores de tamaño variable se debe utilizar POST o PUT

Buenas prácticas en REST

- Usar correctamente los métodos y códigos de error HTTP.
- Para buscar, ordenar, filtrar, paginar, etc. No se crearan nuevos recursos sino que se añadirán los parámetros al GET

```
GET http://example.com/api/books?sort=rank_asc
GET http://example.com/api/books?category=children
...
```

- En una API siempre se debe mantener la compatibilidad hacia atrás, si no se hace, se debe cambiar el nombre

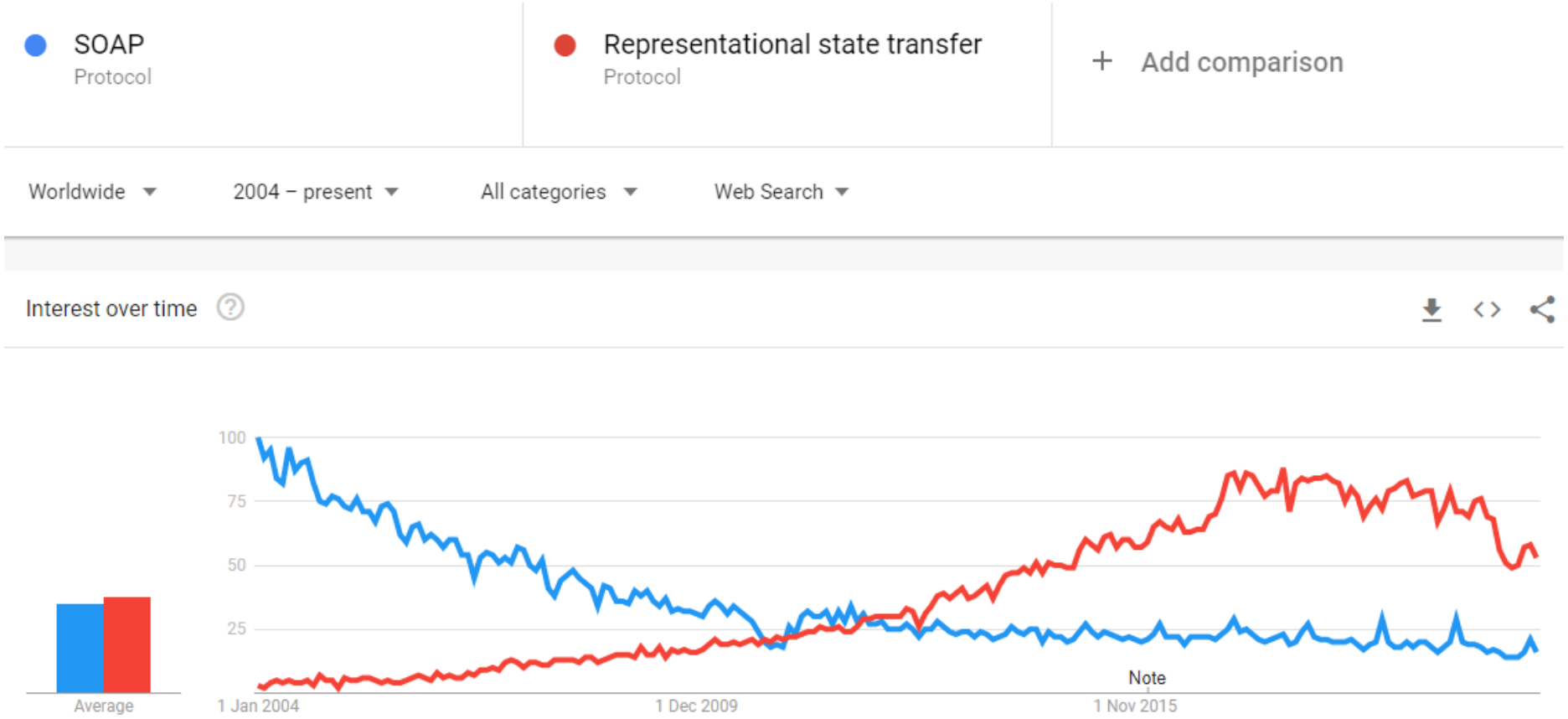
Buenas prácticas en REST

- Usar un mapeo de URIs lo más intuitivo posible y no cambiarlo
- Pensar muy bien cómo va a realizar el consumidor todas las operaciones necesarias
- Aplicar HATEOAS

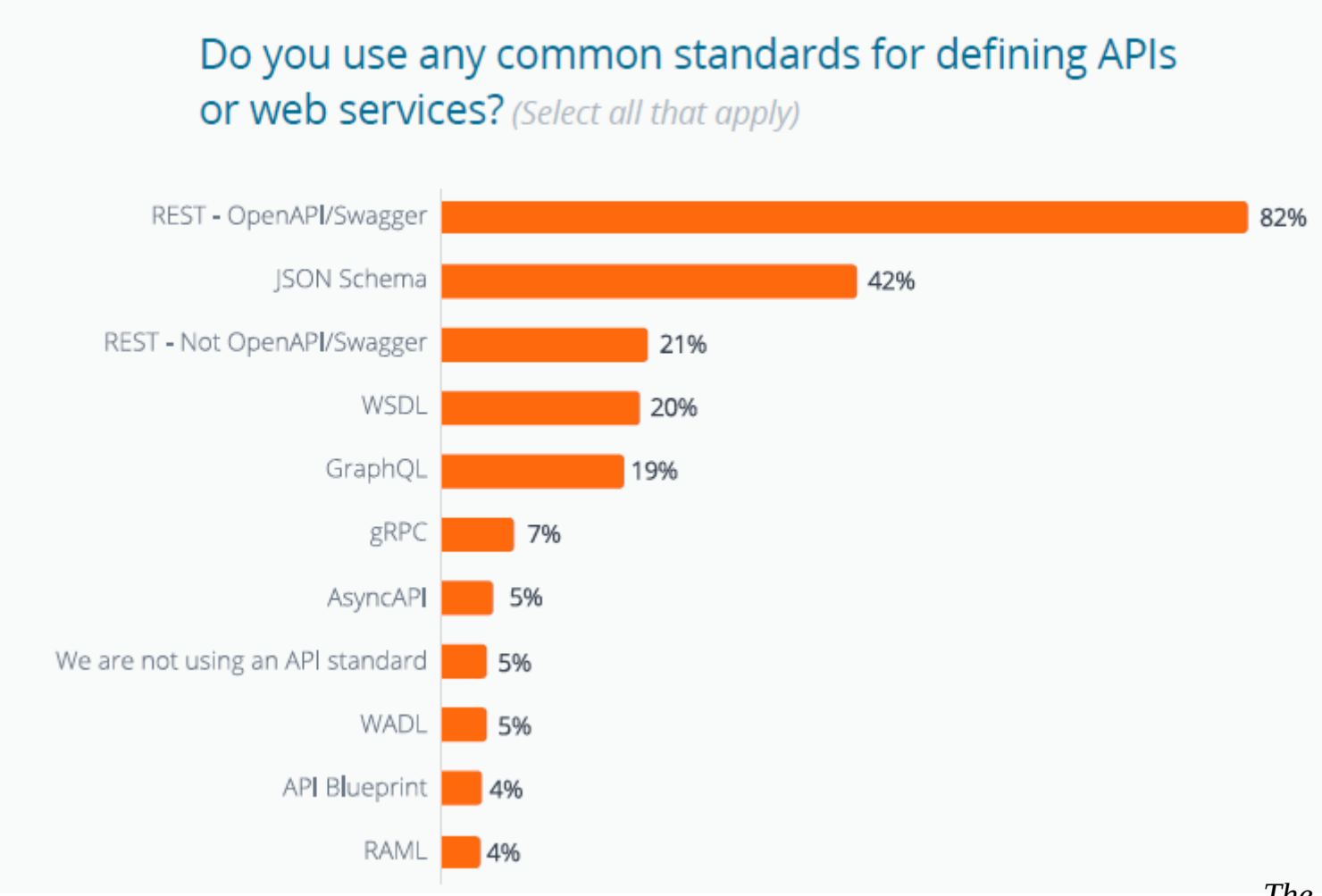
Estandarización

- Lenguajes de descripción (Description Languajes)
- Ejemplos:
  - WADL
  - WSDL
  - OpenAPI (antes Swagger)
  - RAML
- Más información

Simplicity wins (again)

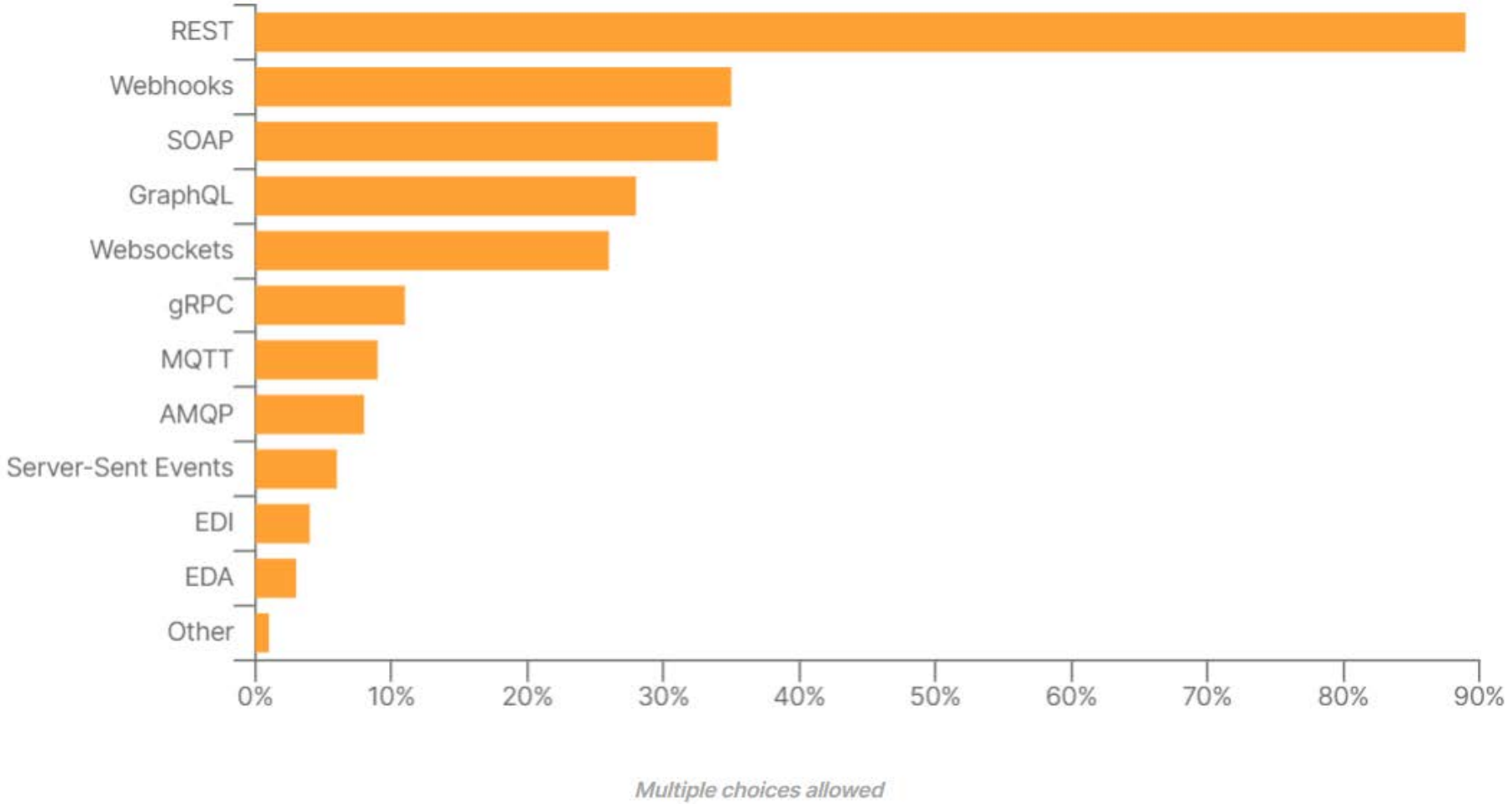


Simplicity wins (again)



SmartBear
The State of API Report 2020
1500+ participants

Simplicity wins (again)



Postman
State of API 2022
37000+ participants

Relaxation

Clearly, it's time to start messing with minds

- REST is not the only architectural style
- My dissertation is about Principled Design, not the one true architecture

What do constraints really mean?

- codify a design choice at the level of architecture
  - to induce certain (good) architectural properties
  - at the expense of certain (bad) trade-offs

What if we relax a given constraint?

- Is it really the end of the world?
- Should waka have its own style?

Conclusion

Use your brains!

- don't design-by-buzzword
- don't believe everything you read
- always keep in mind that **change is inevitable**

Use principled design

- identify desired architectural properties
- select proven architectural styles where appropriate
- constrain behavior to induce properties
- compensate for the design trade-offs


If you get tired, learn to rest, not to quit.

Ejercicio

Tenemos un cine y queremos modernizarnos.

Queremos que la información de la cartelera y las sesiones esté accesible. Actualmente está disponible en nuestra página web, pero queremos que otras aplicaciones puedan trabajar con esa información de forma sencilla obteniéndola de nuestro servicio web para conseguir visibilidad. Aprovechando esta oportunidad queremos también utilizar este sistema para poder trabajar nosotros él.

¿Nos podrías ayudar?

Ejercicio

- Diseña un servicio web basado en REST
- Define las diferentes rutas que necesitamos
- Piensa en las diferentes acciones que se pueden realizar
- Piensa en ejemplos de mensaje y los campos que deberían tener
- Considera los códigos de estado

Referencias

- Tesis de Roy Fielding

  https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Charlas de Roy Fielding

  https://www.slideshare.net/royfielding
- Charla Alexei Skachykkin

  https://www.slideshare.net/alexeiskachykhin/representational-state-transfer-36518469
- Richardson Maturity Model

  https://martinfowler.com/articles/richardsonMaturityModel.html

YAML

YAML

- YAML Ain't Markup Language
- Data Serialization Language
  - Representación de objetos o estructura de datos
  - Formato para almacenar o enviar
- Machine and human readable
- Usado principalmente como lenguaje de configuración
- Desde 2001
- Versión 1.2.2 en octubre de 2021
  - Superset de JSON

YAML

```
blueprint:
  name: Wake-up light alarm with sunrise effect
  description: >
    A wake-up light alarm with a brightness and color temperature sunrise
    effect. Note: Requires date_time_iso sensor in configuration, not manually executable!
  domain: automation
  input:
    light_entity:
      name: Wake-up light entity
      description: The light to control. Turning it off during the sunrise will keep
        it off. Color temperature range is auto-detected.
      selector:
        entity:
          domain: light
action:
- choose: []
  default: !input 'pre_sunrise_actions'
mode: single
max_exceeded: silent
```

YAML

- Comentarios con # y hasta el final de la línea
  - No hay comentarios multilinea
- Puede haber varios elementos raíz
- Indentación:
  - Al menos un espacio (recomendado dos)
  - Mismo número de espacios para todo el nivel
  - No se puede usar el tabulador
- Comienzo del documento: ---
- Final del documento: ...

YAML

Tipos de datos de los nodos:

- Escalares (datos atómicos):
  - Strings: no requieren comillas
  - Números
  - Booleans
  - `null`
- Mapping: pares clave-valor
- Secuencia: lista de nodos

YAML

Mapping

- Pares clave-valor separados por dos puntos y espacio (:)
- Cada par en una línea
- Si no se indica un valor se considera null
- Las claves no se pueden repetir
- Las claves pueden tener cualquier contenido

```
street: Santa Claus Lane
zip code:
city: North Pole
```

YAML

Secuencia:

- Lista de nodos
- Cada uno empieza por guion y espacio (-)
  - Cuenta como indentación
- Cada uno en su propia línea

```
autores:
- Oren Ben-Kiki
- Clark Evans
```

YAML

Secuencia:

- Se pueden anidar

```
- - 2
  - 3
- - 3
  - 6
```

YAML

Alias y anchor

- Permite reutilizar información
- anchor: `&nombreAlias`
- alias: `*nombreAlias`

```
shipping address: &address      #   Anchor
  street: Santa Claus Lane      # ┐
  zip: 12345                    # │ Anchor content
  city: North Pole              # ┘
billing address: *address       #   Alias
```

YAML

String

- Plain
  - Sin comillas
  - No se pueden usar caracteres de escape
  - No se permiten ciertas combinaciones de caracteres:
    - `:<space>`
    - `<space>#`
  - No se permiten ciertos caracteres al principio

YAML

String

- Single quote '
  - Cualquier carácter excepto ' es interpretado literalmente
  - La comilla se puede escapar con dos comillas simples: ''
  - Se puede extender en múltiples líneas

YAML

String

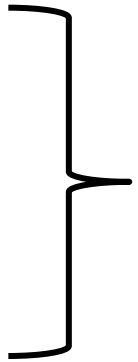
- Double quote "
  - Se pueden usar caracteres de escape (lista): `\n, \t`
  - El salto de línea se reemplaza por un espacio
    - Una `\` al final de una línea se puede usar para que no se añada el espacio
  - Por lo demás igual que las comillas simples

YAML

|

- Literal block scalar
- Referencia las siguientes líneas con la misma indexación
- Todas las líneas son un bloque
- Un salto de línea implica un salto de línea

```
run: |
  ./configure
  make
  make test
```



```
run: "./configure\nmake\nmake test\n"
```

YAML

>

- Folded block scalar
- Referencia las siguientes líneas con la misma indexación
- Todas las líneas son un bloque
- Un salto de línea implica un espacio
- Dos salto de línea implican un salto de línea
  - Cada salto extra es otro salto de línea
- Se conservan los espacios y tabuladores al final de cada línea

YAML

```
---
text: >
  this is the first
  long line

  and this is the
  second
---
text: "this is the first long line\nand this is the second\n"
```

YAML

Flow Style

- Se pueden usar [ ] para las secuencias
- Cada entrada se separa con comas
- Caracteres extra no válidos en plain strings: []{},

```
perl:
- 5.8
- 5.10
- 5.12
- 5.14
- 5.16
```

```
perl: [5.8, 5.10, 5.12, 5.14, 5.16]
```

YAML

Flow Style

- Se pueden usar {} para los mappings
- Cada entrada se separa con comas
- Caracteres extra no válidos en plain strings: []{},

```
- x: 3
  y: 4
  z: 5
- x: 5
  y: 4
  z: 3
```

```
- {x: 3, y: 4, z: 5}
- {x: 5, y: 4, z: 3}
```

YAML

- ¿A qué equivalen?

```
---
data:
- step
- - level
---
rolls:
- - 2
  - 5
```

YAML

- ¿A qué equivalen?

```
---
sequence:
- a: b
  c: d
---
data:
- a: b
  - c: d
```

YAML

- ¿Es inválido?

```
features:
- name: lorem ipsum
  bullets: |
test:
- name: >
    lorem
      ipsum 2
  bullets:
test:
  acl: loopback
  acl: admin
```

YAML – Referencias

- Información: https://yaml.org/
- Tutorial: https://www.yaml.info/index.html
- YAML Data Project: https://yaml.com/
- Validator: https://jsonformatter.org/yaml-validator

OPENAPI

OpenAPI

- Estándar para la descripción de APIs
- Desde 2010
- Anteriormente Swagger
- Versión más reciente v3.1.0 (febrero 2021)
- Gestionada por la Open API Initiative bajo la Linux Foundation
- Fundadores: Google, IBM, Microsoft, SmartBear…
- Documentación
- Especificación (OAS)
- Validador

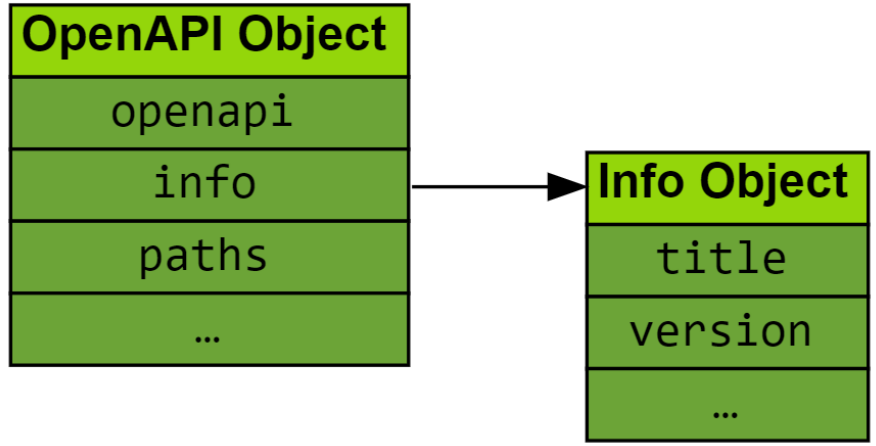
OpenAPI – Ejemplo

```
openapi: 3.1.0
info:
  title: A minimal OpenAPI document
  version: 0.0.1
  description: OpenAPI document example
paths: {} # No endpoints defined
```

OpenAPI

OpenAPI Object

- `openapi` (string): versión de OAS. REQ
- `info` (Info Object): información general. REQ
  - `title` (string): nombre de la API. REQ
  - `version` (string) versión de la API. REQ
  - description (string)
- `paths` (Paths Object):
  - descripción de los endpoints
  - Información de parámetros y respuestas



OpenAPI

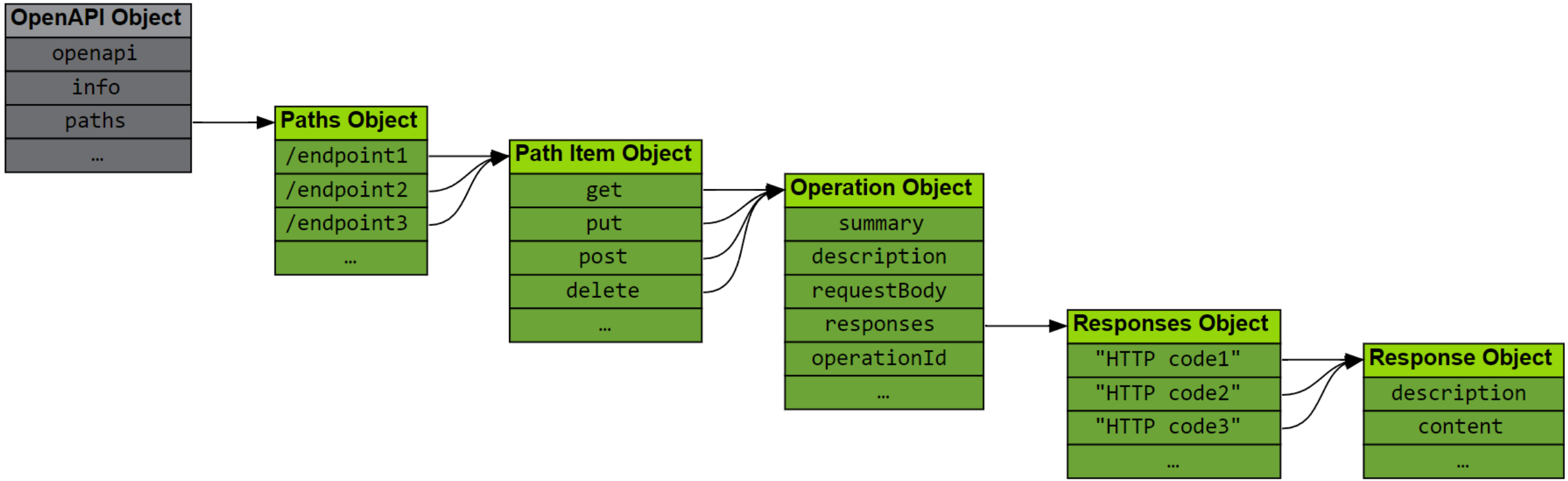
OpenAPI Object

- `servers:` Array de Server Objects que indican la ruta
- `webhooks:` Información de los webhooks
- `security`: Información sobre los mecanismos de seguridad
- `components`: Definir schemas parámetros y otros elementos para reutilizar en el documento
- Y más opciones

OpenAPI – Ejemplo

```
…
paths:
  /board:
    get:
      summary: Get the whole board
      description: Retrieves the current state of the board and the winner.
      responses:
        "200":
          description: "OK"
          content:
            …
```

OpenAPI


OpenAPI Object
openapi
info
paths
…
Paths Object
/endpoint1
/endpoint2
/endpoint3
…
Path Item Object
get
put
post
delete
…
Operation Object
summary
description
requestBody
responses
operationId
…
Responses Object
"HTTP code1"
"HTTP code2"
"HTTP code3"
…
Response Object
description
content
…

OpenAPI

Paths Object

- Cada una de las rutas empezando por /
- Puede contener un template entre { y }
  - Tiene que estar definido después

```
/books/{id}
```

  - No puede haber dos templated paths con la misma jerarquía aunque tengan nombres diferentes

```
/books/{id}
/books/{ref}
```

OpenAPI

Paths Object

- Puede contener un template entre { y }
  - Puede haber ambigüedades

```
/{entity}/me
/books/{id}
```

- Contiene un Path Item Object

OpenAPI

Path Item Object:

- Listado de los métodos soportados:
  - `get`
  - `put`
  - `post`
  - `delete`
  - `Y más…`
- Contiene un Operation Object

OpenAPI

Operation Object:

- `summary`
- `description`
- `parameters` (Parameter Object): lista de parámetros
- `requestBody` (Request Body Object)
- `responses` (Responses Object):
  - lista de posibles respuestas
  - Al menos tiene que haber una y debería ser la de éxito

OpenAPI

Responses Object

- Código de estado HTTP
- Entre comillas
- Se pueden usar placeholders: "1XX", "2XX", "3XX", "4XX" y "5XX"
  - Tiene precedencia un método específico a un placeholder
- Contiene un Response Object

OpenAPI

Response Object:

- description (string): REQ
- content (Media Map): listado de Media Types

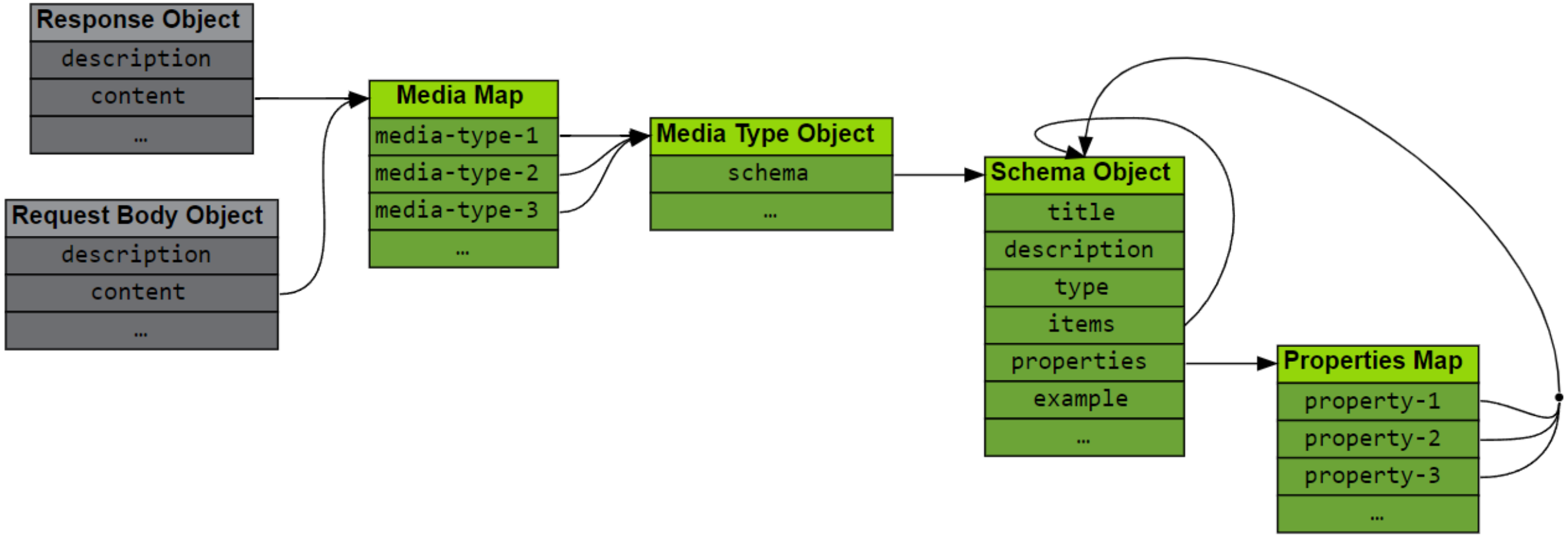
OpenAPI – Ejemplo

```
(…)
content:
  application/json:
    schema:
      type: string
      enum:
      - Alice
      - Bob
      - Carl
```

```
(…)
content:
  application/json:
    schema:
      type: array
      minItems: 1
      maxItems: 10
      items:
        type: integer
```

```
(…)
content:
  application/json:
    schema:
      type: object
      properties:
        productName:
          type: string
        productPrice:
          type: number
```

OpenAPI


Response Object
description
content
…
Request Body Object
description
content
…
Media Map
media-type-1
media-type-2
media-type-3
…
Media Type Object
schema
…
Schema Object
title
description
type
items
properties
example
…
Properties Map
property-1
property-2
property-3
…

OpenAPI

Media Map

- MIME types (RFC 6838)
- Ejemplos:
  - text/plain
  - application/xml
  - application/json
  - multipart/form-data
- Contiene un Media Type Object

OpenAPI

Media Type Object

- `schema` (Schema Object)
- `example`
- `examples`

} Sólo puede estar uno de los dos

Schema Object

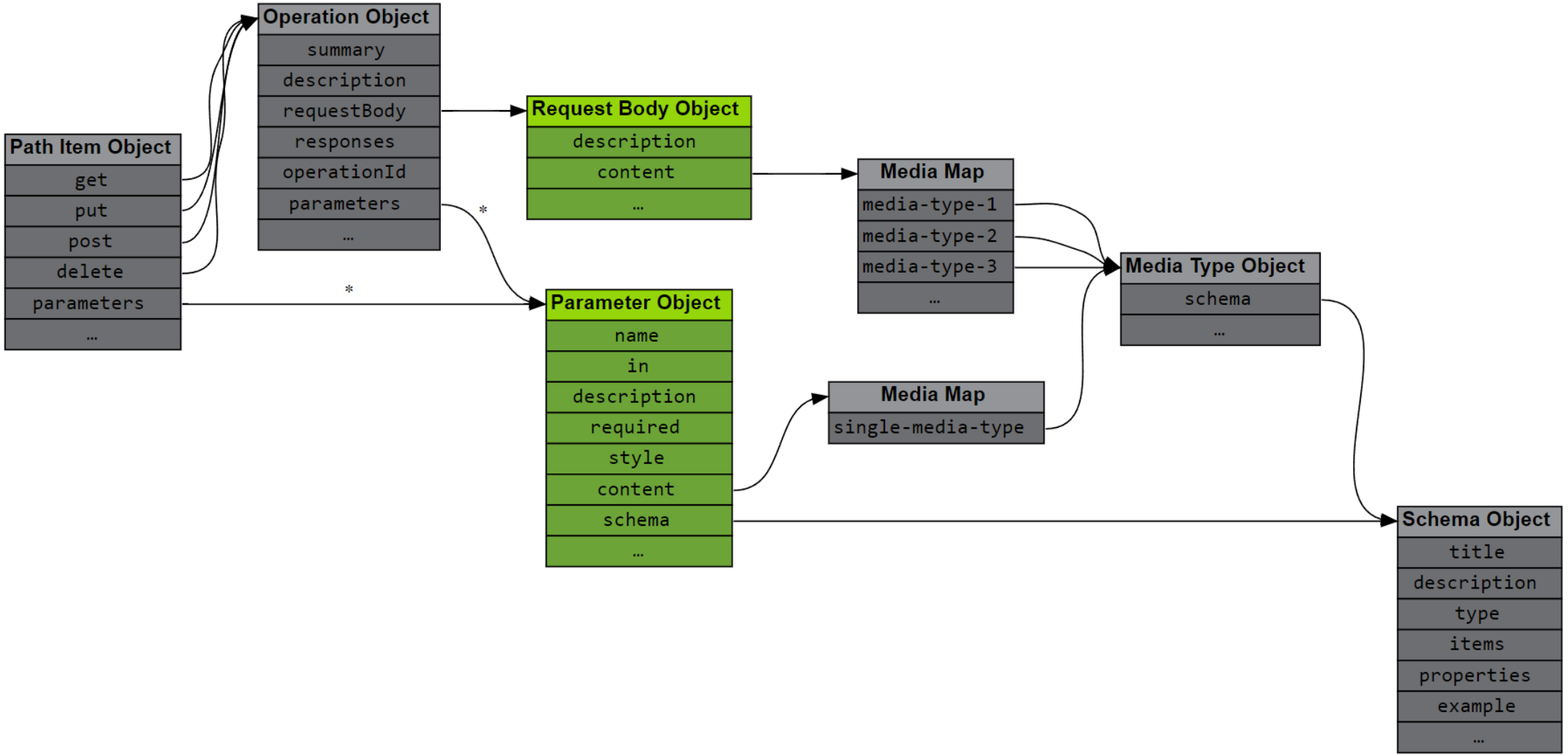
- Superset de JSON Schema 2020-12

OpenAPI

```
#Ejemplo 1
(…)
paths:
  /users/{id}:
    get:
      parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
```

```
#Ejemplo 2
(…)
parameters:
- name: id
  in: query
  schema:
    type: integer
    minimum: 1
    maximum: 100
```

OpenAPI


Path Item Object
get
put
post
delete
parameters
…
Operation Object
summary
description
requestBody
responses
operationId
parameters
…
Request Body Object
description
content
…
Media Map
media-type-1
media-type-2
media-type-3
…
Media Type Object
schema
…
Parameter Object
name
in
description
required
style
content
schema
…
Media Map
single-media-type
Schema Object
title
description
type
items
properties
example
…
*
*

OpenAPI

Parameter Object

- `name`: único en cada localización. REQ
- `in`: Localización del parámetro (path, query …). REQ
- `description`
- `required` (boolean):
  - false por defecto
  - REQ. para los path parameters y tiene que ser `true`

OpenAPI

Parameter Object

- `style`: cómo se va a serializar el parámetro
- `explode` (boolean)

style:	simple	form	label	matrix
Primitive type	1234	id=1234	.1234	;id=1234
Array (explode=false)	1,2,3	ids=1,2,3	.1.2.3	;ids=1,2,3
Array (explode=true)	1,2,3	ids=1&ids=2&ids=3	.1.2.3	;ids=1;ids=2;ids=3
Object (explode=false)	R,1,G,2,B,3	color=R,1,G,2,B,3	.R.1.G.2.B.3	;color=R,1,G,2,B,3
Object (explode=true)	R=1,G=2,B=3	R=1&G=2&B=3	.R=1.G=2.B=3	;R=1;G=2;B=3

OpenAPI

```
(…)
requestBody:
  content:
    application/json:
      schema:
        type: integer
        minimum: 1
        maximum: 100
```

OpenAPI

Request Body Object

- content (Media Map): REQ.

OpenAPI – Ejemplo

Ejemplo final

```
openapi: 3.1.0
info:
  title: Tic Tac Toe
  description: |
    This API allows writing down marks on a Tic Tac Toe board
    and requesting the state of the board or of individual squares.
  version: 1.0.0
tags:
  - name: Gameplay
paths:
  # Whole board operations
  /board:
    get:
      summary: Get the whole board
      description: Retrieves the current state of the board and the winner.
      tags:
        - Gameplay
      operationId: get-board
      responses: (…)
```

OpenAPI – Ventajas

- Description Validation and Linting:
  - Description file syntactically correct
  - Adheres to a specific version of the Specification
  - Team’s formatting guidelines
- Data Validation:
  - Data flowing through your API (in both directions) is correct
  - During development and once deployed

OpenAPI – Ventajas

- Documentation Generation:
  - Create traditional human-readable documentation
  - Always stays up-to-date
- Code Generation:
  - Server and client code in any programming language
- Graphical Editors:
  - Creation of description files using a GUI

OpenAPI – Ventajas

- Mock Servers:
  - Fake servers for testing
  - Before you write a single line of code
- Security Analysis:
  - Discover possible vulnerabilities at the API design stage

OpenAPI – Referencias

- Documentación
- OpenAPI Specification v3.1.0
- OpenAPI diferencias entre v3.1.0 y v3.0.3
- OpenAPI Map
- OpenAPI Implementations
- OpenAPI Tools
- PetStore
- https://schema.org/
- Validador del schema online (sólo hasta versión 3.0.3)
- Diferencias entre Swagger y OpenAPI

OpenAPI – Ejercicio

- Genera el documento OpenAPI para el ejercicio del cine

Ejercicio

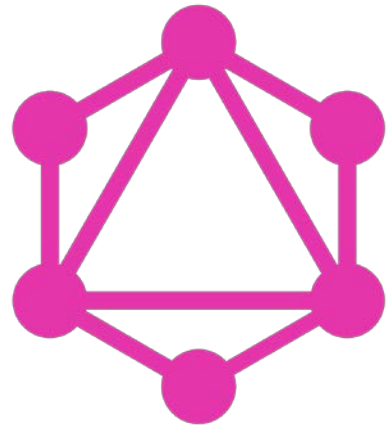
- Diseña un servicio web basado en REST
- Define las diferentes rutas que necesitamos
- Piensa en las diferentes acciones que se pueden realizar
- Piensa en ejemplos de mensaje y los campos que deberían tener
- Considera los códigos de estado

OpenAPI – Swagger

- Versión del standard anterior a OpenAPI
- Herramientas para OpenAPI
- Documentación
- Mock Servers
- Clientes
- Generación
- https://swagger.io/
- Swagger Inspector

OTRAS ALTERNATIVAS

GraphQL



- Desarrollado por Facebook desde 2012
- Primera versión pública en 2015
- Estable desde 2018
- Gestionado en la actualidad por la Linux Foundation
- Permite solicitar exactamente los recursos necesarios
- Desventajas:
  - Complica el cacheado
  - Complejo para servicios sencillos
- Más información

GraphQL

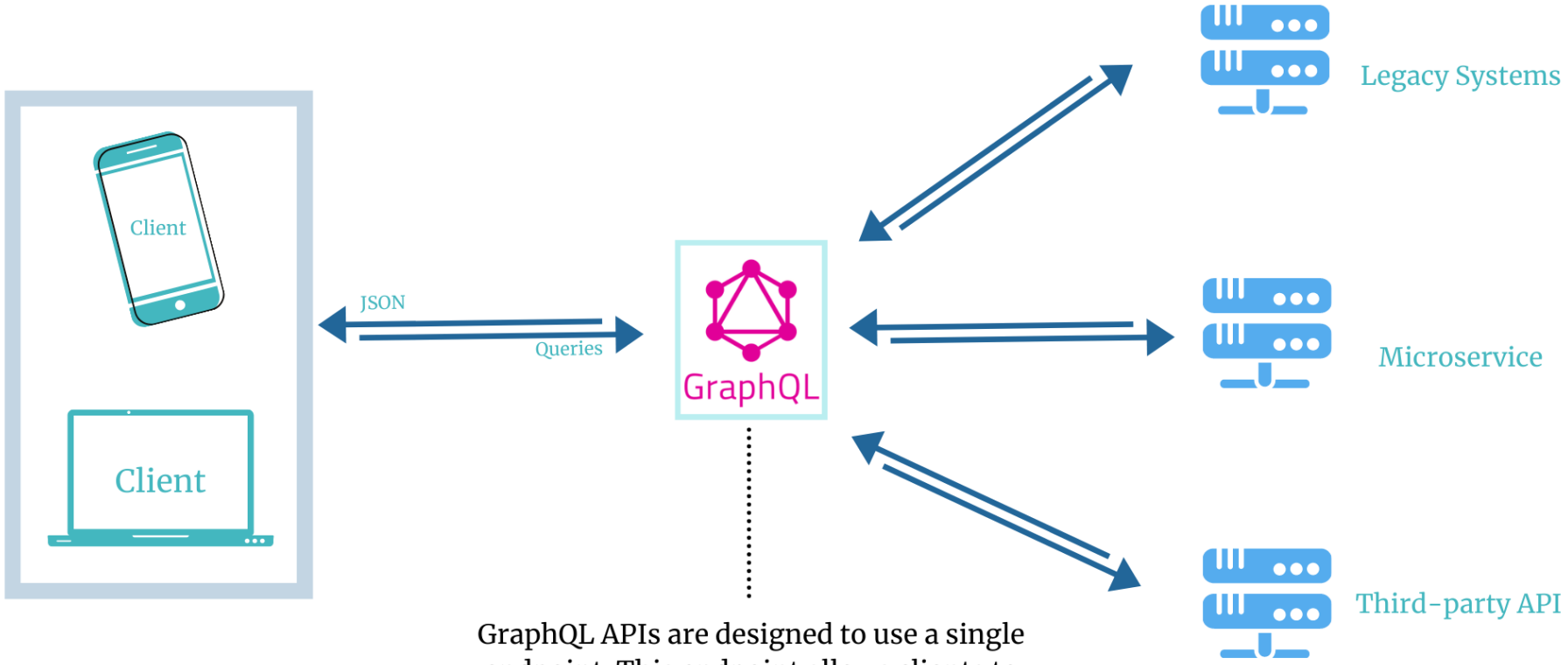
REQUEST

```
{
    orders {
        id
        productsList {
            product {
                name
                price
            }
            quantity
        }
        totalAmount
    }
}
```

RESPONSE


```
{
  "data": {
    "orders": [
      {
        "id": 1,
        "productsList": [
          {
            "product": {
              "name": "orange",
              "price": 1.5
            },
            "quantity": 100
          }
        ],
        "totalAmount": 150
      }
    ]
  }
}
```

GraphQL



GraphQL APIs are designed to use a single endpoint. This endpoint allows clients to reach different types of data storage.

GraphQL

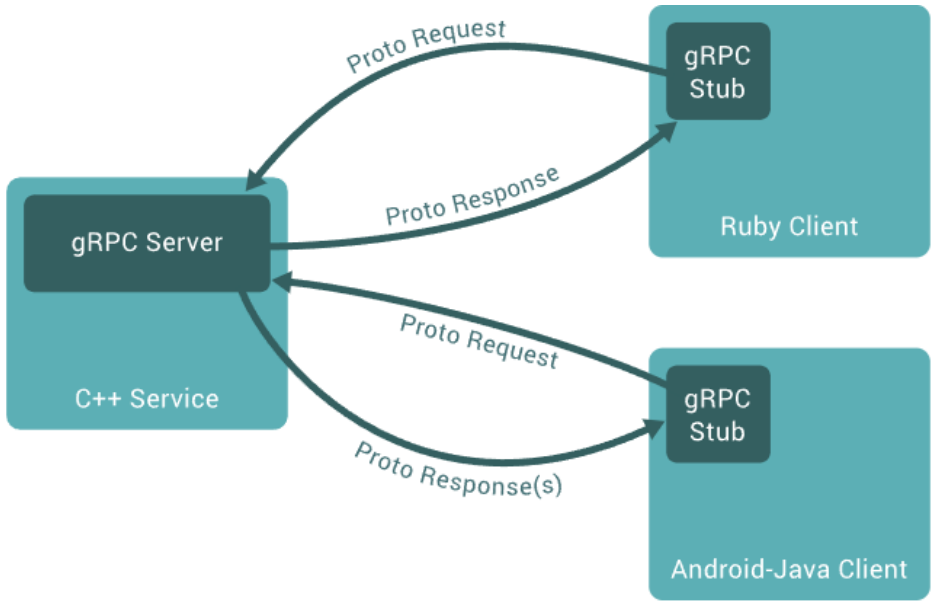
- Web
- Tutorial Interactivo
- Github GraphQL API
  - Anuncio
- Curso

gRPC

- Google Remote Procedure Call
- Open source
- HTTP/2 para transporte
  - Imposible crear un cliente para el navegador
- Más información

gRPC

- Invocas métodos remotos como si estuvieran en local



gRPC

- Por defecto usa Protocol Buffers (Protobuf) como formato de datos
  - Parecido a JSON

```
message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}
```

Comparativa

	REST	RPC	GraphQL
What?	Exposes data as resources and uses standard HTTP methods to represent CRUD operations	Exposes action-based API methods. Clients pass method name and arguments	A query language for APIs. Clients define the structure of the response
Example services	Stripe, GitHub, Twitter, Google	Slack, Flickr	Facebook, GitHub, Yelp
Example usage	`GET /users/<id>`	`GET /users.get?id=<id>`	`query ($id:String!) { user(login: $id){ name company createdAt }}`
HTTP verbs used	GET, POST, PUT, PATCH, DELETE	GET, POST	GET, POST

Comparativa

	REST	RPC	GraphQL
Pros	• Standard method name, arguments format, and status codes • Utilizes HTTP features • Easy to maintain	• Easy to understand • Lightweight payloads • High performance	• Saves multiple round trips • Avoids versioning • Smaller payload size • Strongly typed • Built-in introspection
Cons	• Big payloads • Multiple HTTP round trips	• Discovery is difficult • Limited standardization • Can lead to function explosion	• Requires additional query parsing • Backend performance optimization is difficult • Too complicated for a simple API
When to use?	For APIs doing CRUDlike operations	For APIs exposing several actions	When you need querying flexibility; great for providing querying flexibility and maintaining consistency

Webhooks

- Reverse API
- El servidor hace una petición al cliente
- HTTP Post
- Notificación de eventos
  - No debería incluir payload
- Evita que el cliente tenga que hacer *polling* constante para ver si hay cambios:
  - Solo 1,5% de las llamadas de polling de Zapier tenía nuevos datos

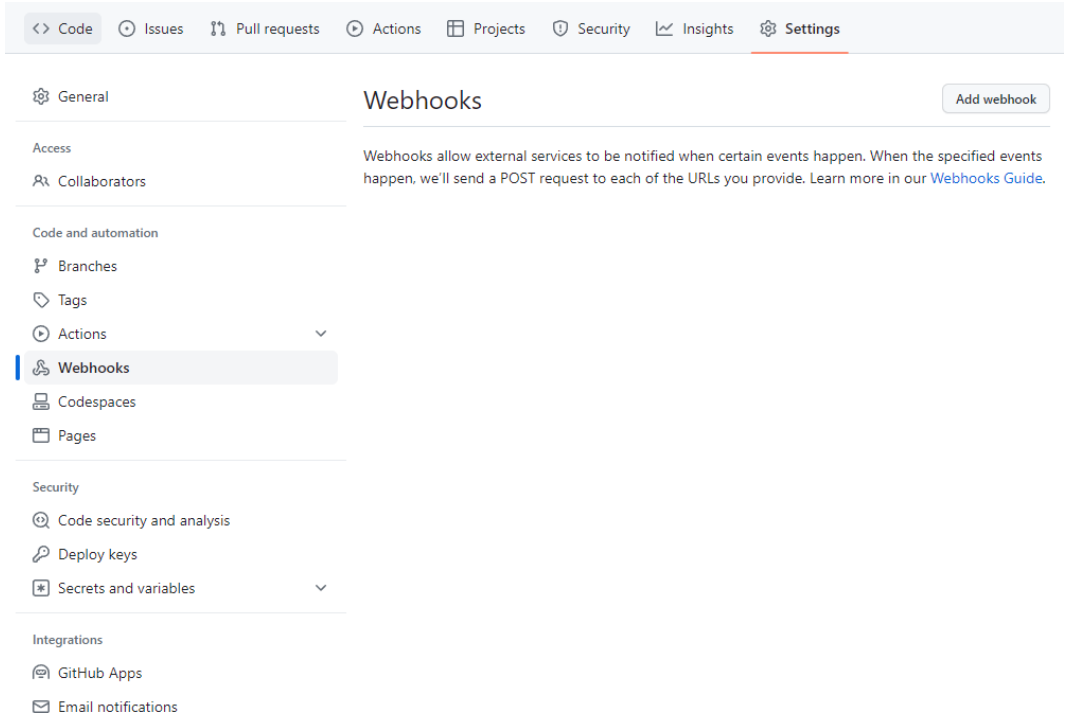
Webhooks

Pasos a seguir:

1. Necesitamos un endpoint para recibir las llamadas
2. Enviamos el endpoint al servidor
3. Se genera nueva información en el servidor
4. El servidor envía un POST a nuestro endpoint

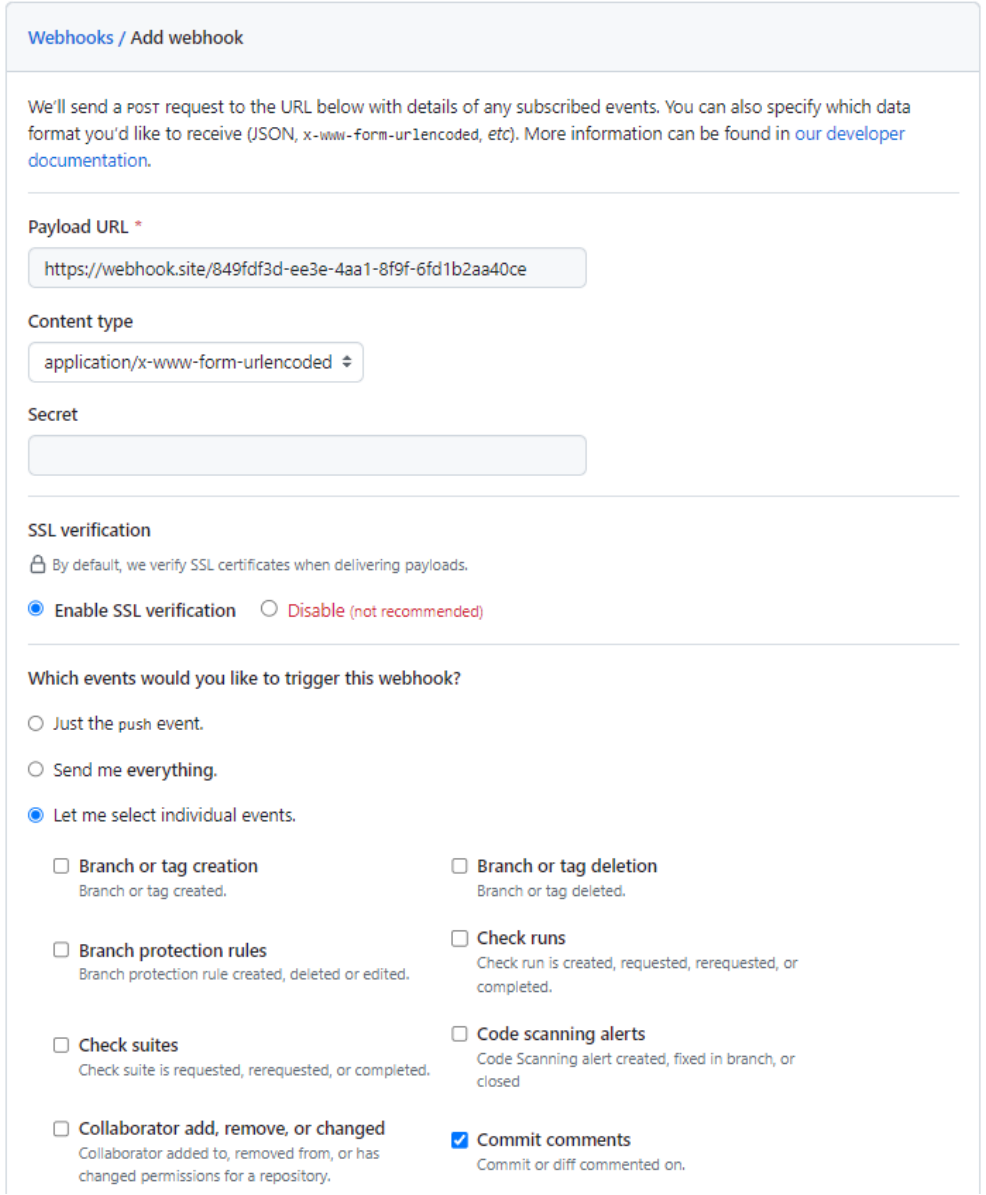
Webhooks - Demo

- Accede a https://webhook.site/
- Vete al proyecto tuyo de Github que quieras
- Entra en Settings > Webhooks



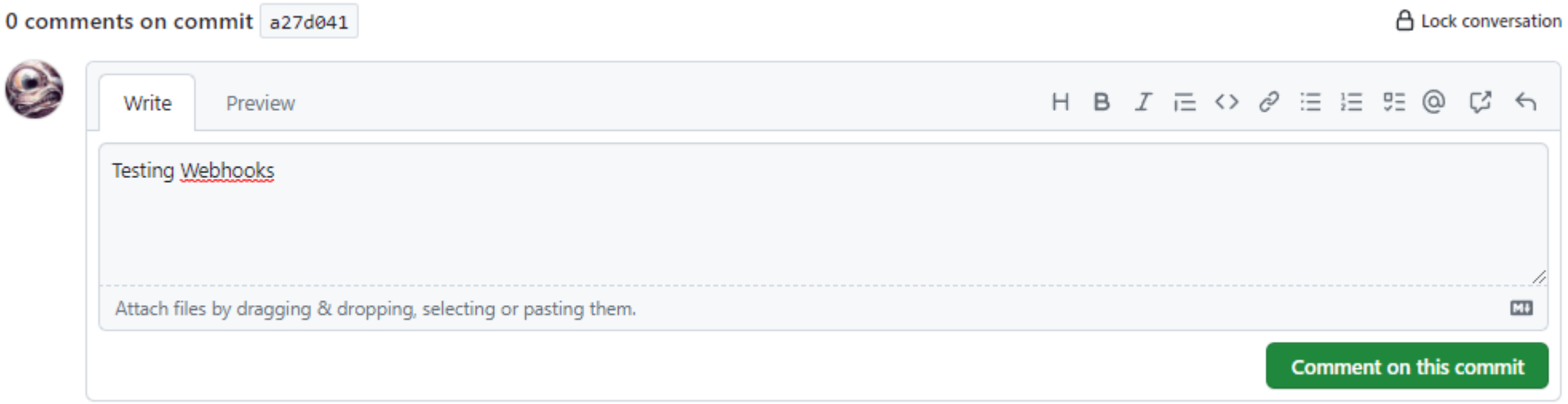
Webhooks - Demo

- Introduce la URL
- Selecciona "Let me select individual events"
- Marca "Commit comments"



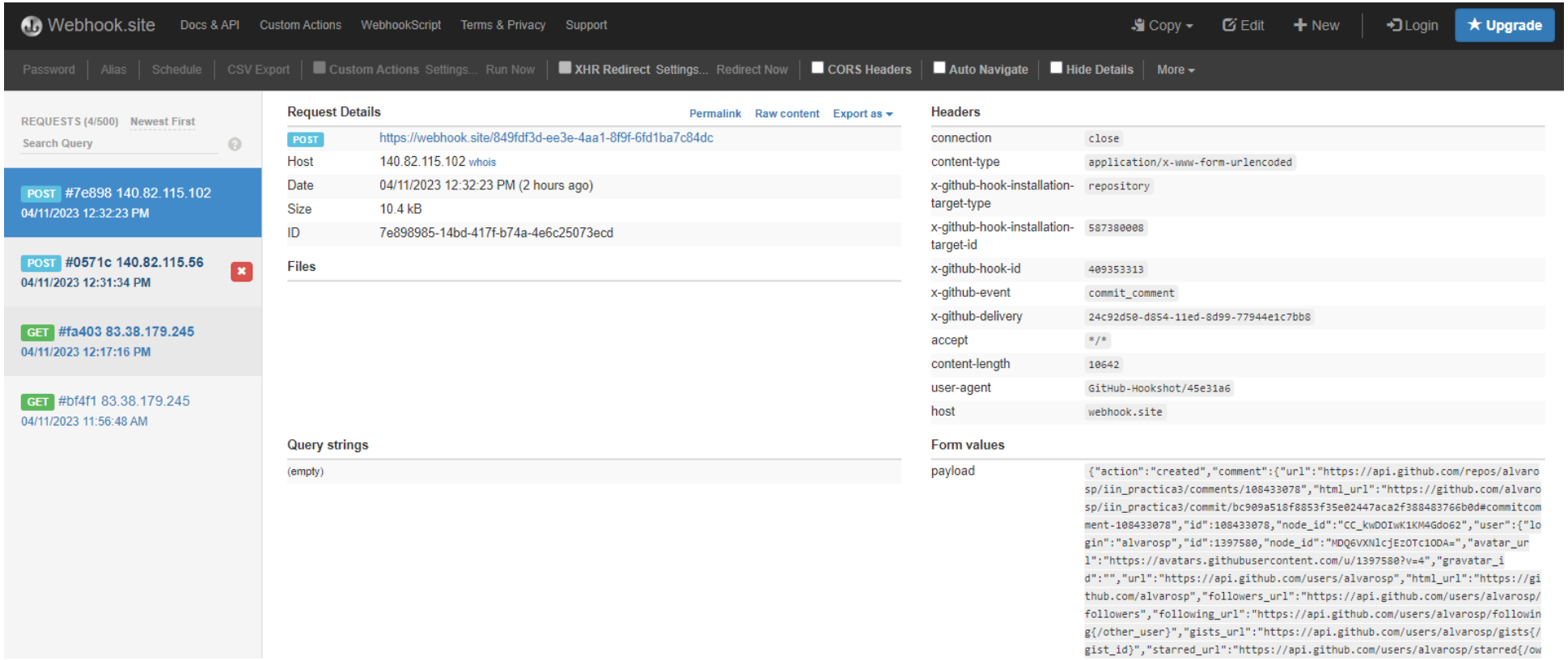
Webhooks - Demo

- Añade un comentario



Webhooks - Demo

- Comprueba que aparece en la interfaz



Webhooks

- Más información
- Github Webhooks

Escalando la API

Scaling Throughput

- Número de llamadas a la API por segundo
- Encontrar los cuellos de botella
  - Recolectar datos de uso
  - Monitorizar la capacidad
  - Ej.: Disk I/O, Network I/O, CPU, Memory

Escalando la API

Scaling Throughput

- Soluciones:
  - Análisis del código (code profiling)
  - Añadir recursos
  - Indexar las DB
  - Cacheado de la información más frecuente
  - Realizando operaciones costosas de forma asíncrona

Escalando la API

Evolving API Design

- Introducing New Data Access Patterns
  - Ej.: solo 1,5% de las llamadas de polling de Zapier tenía nuevos datos
- Adding New API Methods
  - Analizar los casos de uso de los clientes
  - A veces hacen falta menos datos
- Supporting Bulk Endpoints

Escalando la API

Evolving API Design

- Adding New Options to Filter Results:
  - Search filter
  - Date filter
  - Order filter
  - Opciones para indicar si un campo se incluye o no
    - Ejemplo de Twitter

Escalando la API

Paginación

- Evitamos devolver demasiados datos
- Offset-Based Pagination
  - Ej: `https://api.github.com/user/repos?page=5&per_page=10`
  - ¿Qué pasa si hay nuevos datos o se ha eliminado alguno?

Escalando la API

Paginación

- Cursor-Based Pagination
  - `https://api.twitter.com/1.1/followers/ids.json?user_id=12345&count=50` → `{"ids": […], "next_cursor": 1374004777531007833}`
  - `https://api.twitter.com/1.1/followers/ids.json?user_id=12345&count=50&cursor=1374004777531007833`
  - El cursor nos permite recorrer todo hasta que nos devuelva 0
  - Ejemplos de cursor: ID, timestamp, opaque string
  - Más info

Escalando la API

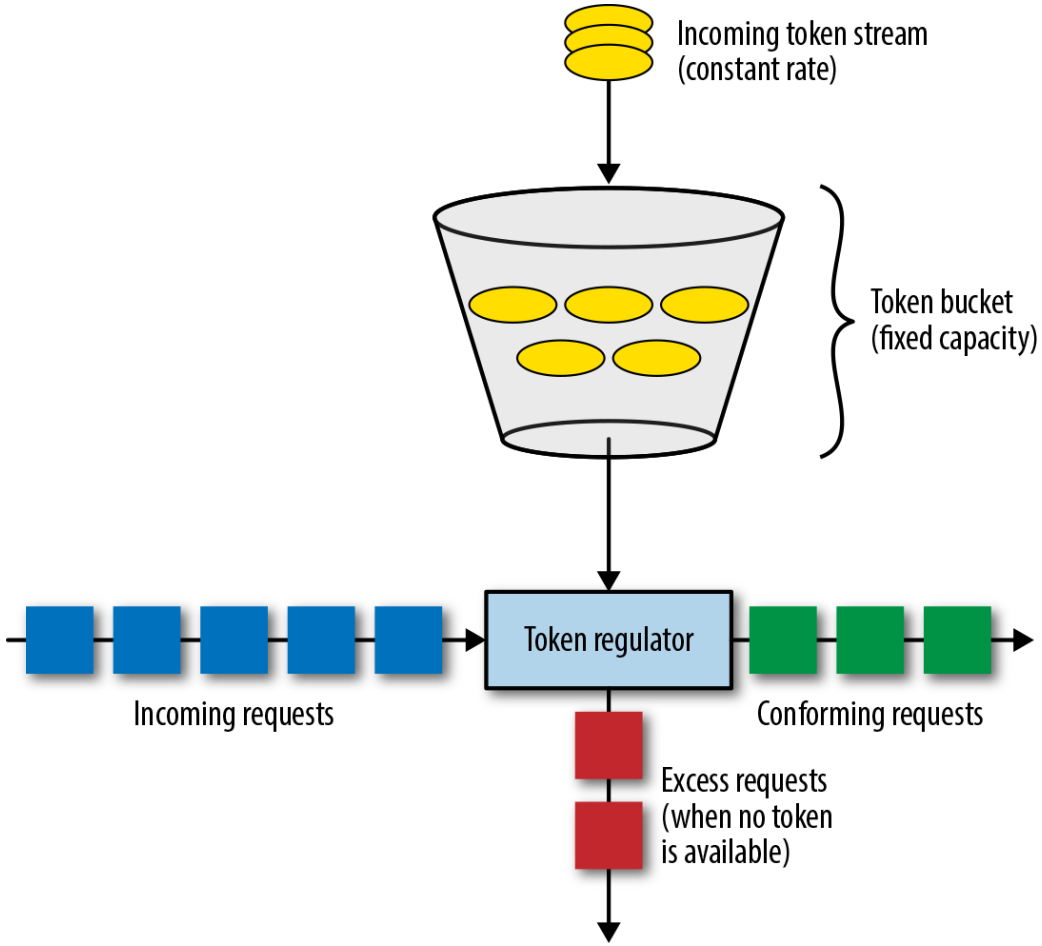
Rate-Limiting APIs

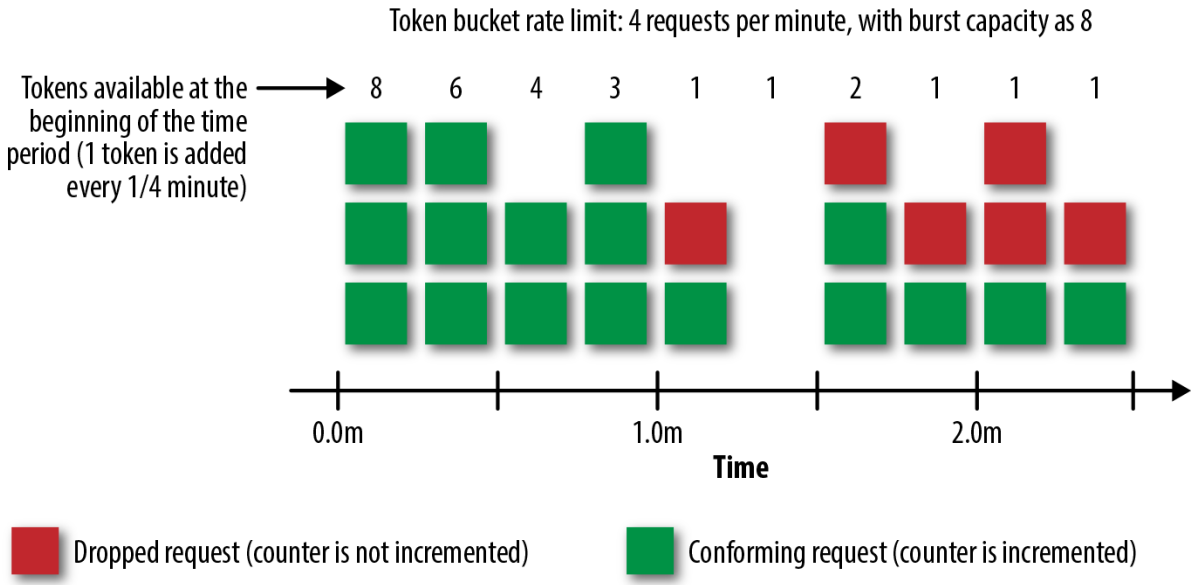
- Limitamos el número de llamadas a nuestra API
- Evitamos ataques DoS y spam
- Podemos permitir límites globales o específicos por endpoints
- Tenemos que medir el tráfico
- Puede que nos interese permitir picos de tráfico
- Quizás tengamos excepciones

Escalando la API

Rate-Limiting APIs

- Token bucket

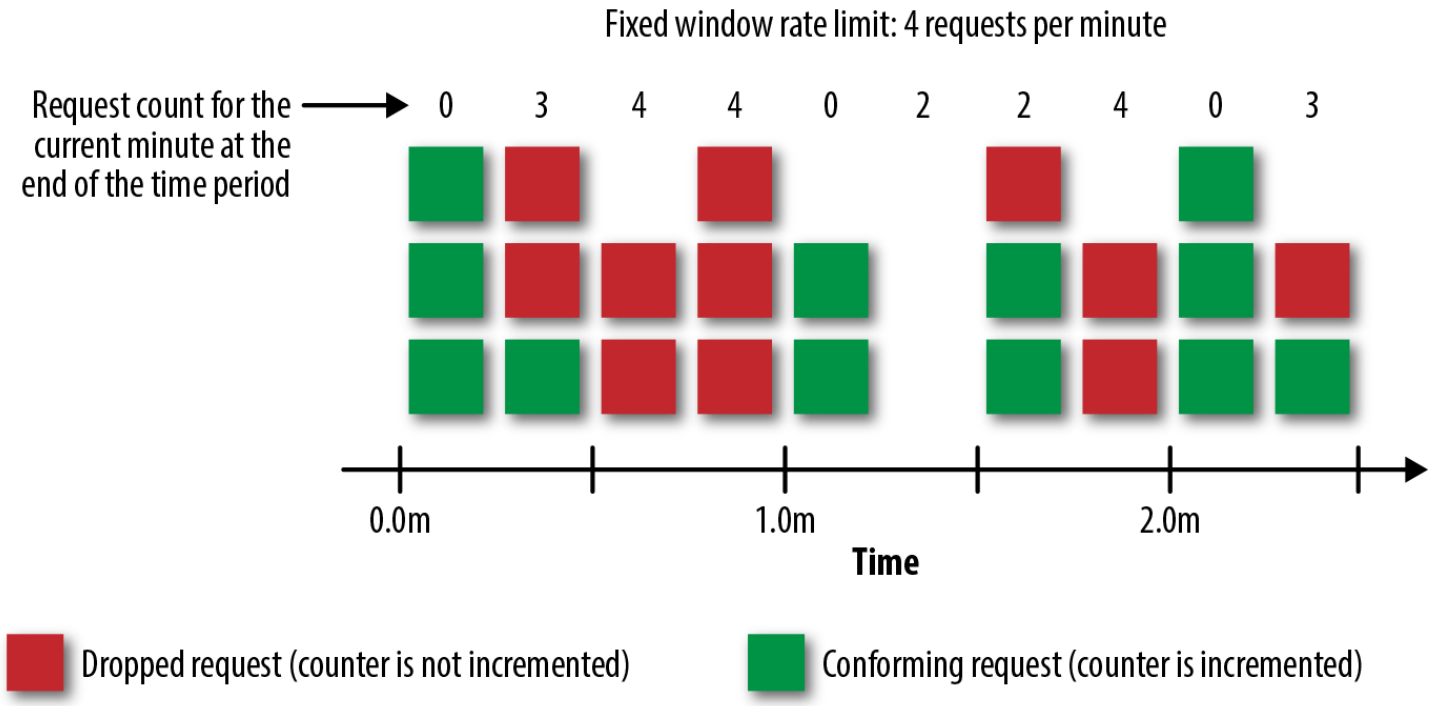




Escalando la API

Rate-Limiting APIs

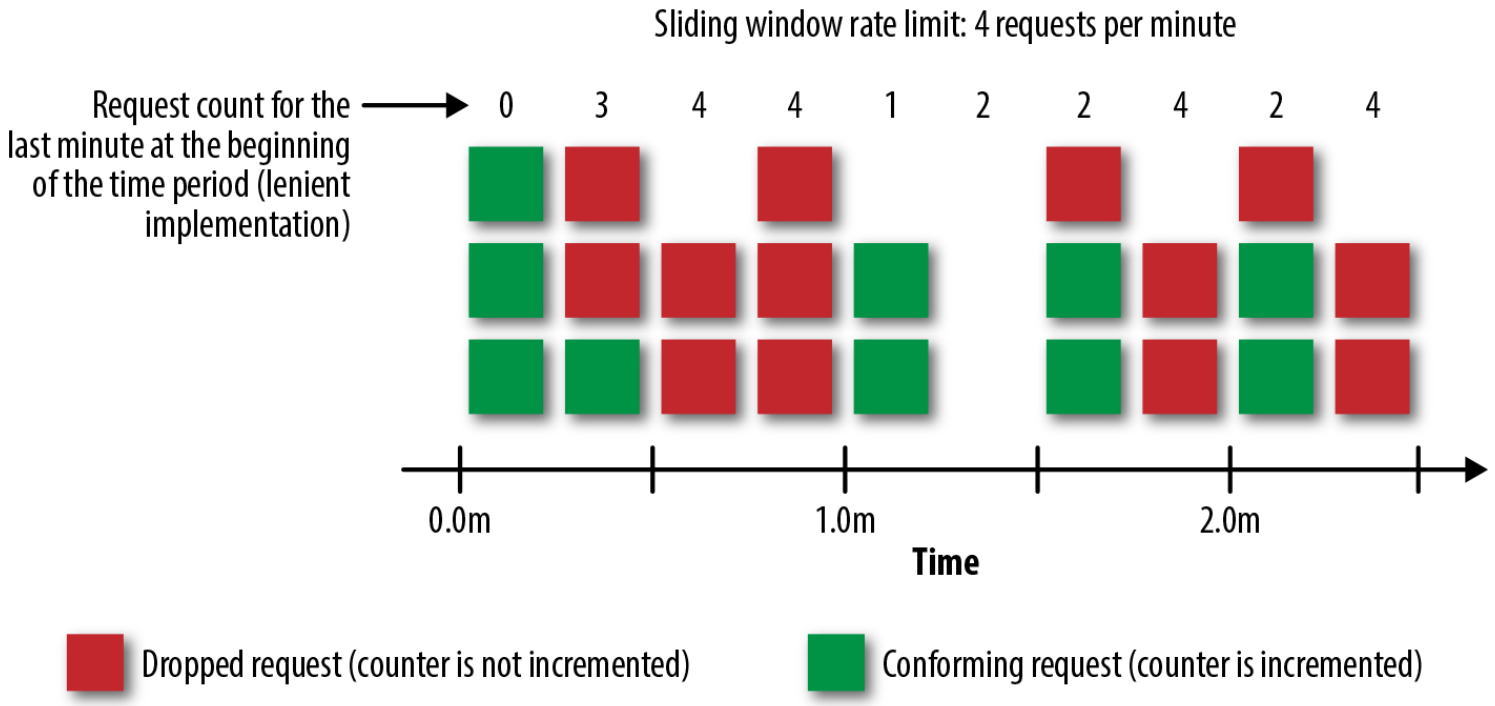
- Fixed-window counter



Escalando la API

Rate-Limiting APIs

- Sliding-window counter



Escalando la API

Rate-Limiting APIs

- Hay que informar al cliente del número de peticiones restantes
  - Indicarlo en las cabeceras
    - `X-RateLimit-Limit`
    - `X-RateLimit-Remaining`
    - `X-RateLimit-Reset`
    - `Ej: curl -I https://api.github.com/users/torvalds`

Escalando la API

Rate-Limiting APIs

- Hay que informar al cliente del número de peticiones restantes
  - Una API en la que pueda consultarlo
  - Tiene que estar en la documentación
  - Código de estado 429 Too Many Requests

```
retry-after: 36
```

Bibliografía

- Restful Web API Patterns and Practices Cookbook; O'Reilly Media; 1st edition; ISBN-13: 978-1098106744
- Designing Web APIs Building APIs That Developers Love; O'Reilly Media; 1st edition; ISBN-13: 978-1492026921
- Pro REST API Development with Node.js; Apress; 1st edition; ISBN-13: 978-1484209189