

Status 100: CONTINUE

Módulo 2: REST en teoría

Repaso

Evolución de la computación

- Programas monolíticos
- Sistemas Operativos
- Bibliotecas / Frameworks
- Inter-process communication (IPC)
- Remote process communication (RPC)

APIs de sistema, de plataforma y de aplicación

Contratos de APIs, economía de las APIs

Motivación para el módulo de hoy

Un quiebre en la evolución de la computación:



Motivación para el módulo de hoy

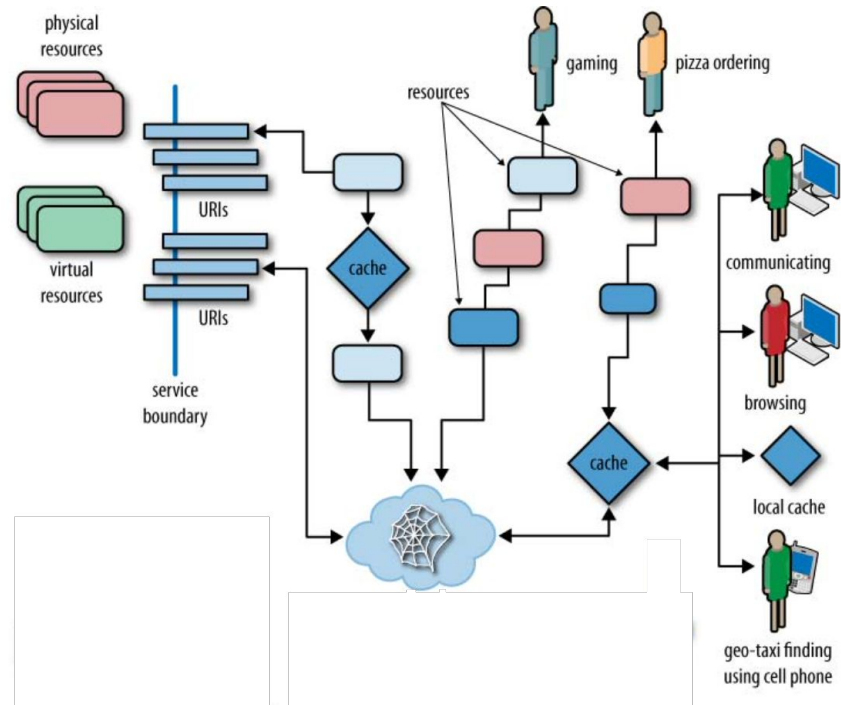
Tim Berners-Lee crea la World Wide Web (WWW) en el CERN en la década de 1990



La World Wide Web: motivación

Crear un sistema para compartir documentos:

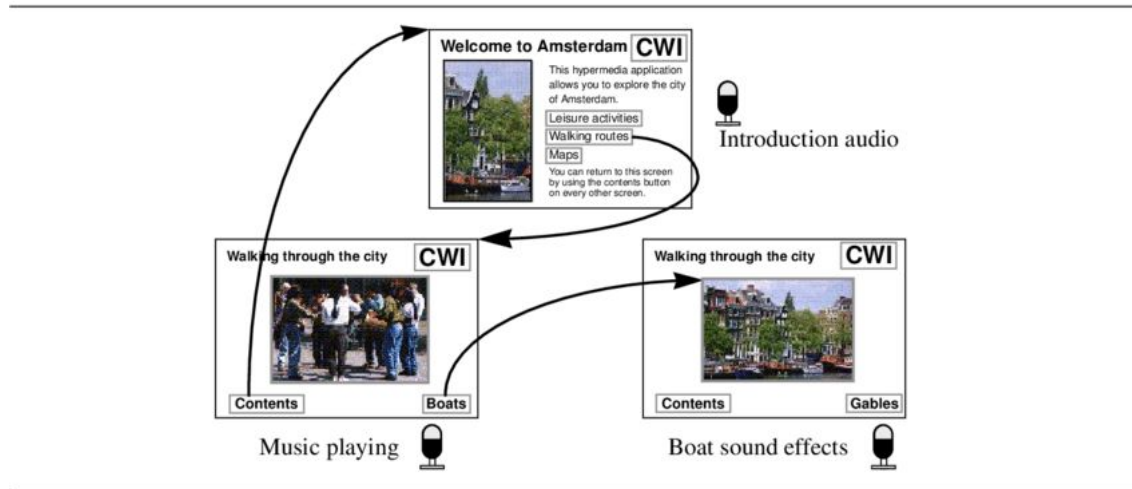
- Con foco en la **facilidad** de uso
- De naturaleza **distribuida**
- Con **bajo acoplamiento**



La World Wide Web: hypermedia

Documentos que hacen referencia a otros documentos

- Explicitar la estructura de grafo que tiene la información
- Más allá de texto: aplica a sonido, video, imágenes

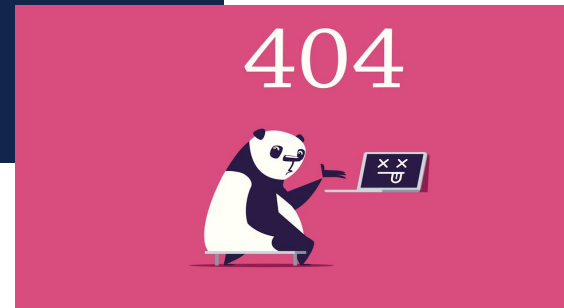
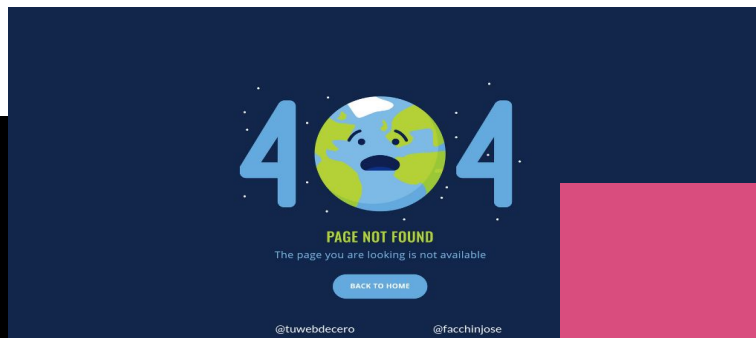
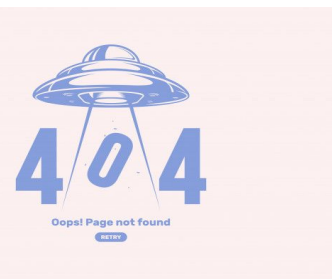


La World Wide Web: hypermedia

Intentos anteriores habían fallado por forzar consistencia en el modelo: **no hay links rotos**

La belleza de la WWW es que habilita el 404 como un código para indicar enlaces rotos

- Filosofía: no hay un ente central



La World Wide Web: recursos

Los **recursos** son los objetos con los que trabajamos al usar la World Wide Web

Pueden ser representaciones digitales de recursos físicos:

- La góndola de las verduras en el super
- El recibo de pago de un café con 2 medialunas de esta mañana

O pueden ser la representación de algún proceso digital:

- Un bot con el que interactuamos para realizar una gestión
- Un proceso para cambiar nuestra contraseña en un sitio

La World Wide Web: recursos y sus identificadores

Un/a **URI** o **Unified Resource Identifier** es el nombre unívoco con el que identificamos a cada recurso.

<https://comprafacil.com.ar/recibos/22874>

<https://comprafacil.com.ar/recibos/ultimo>

<https://comprafacil.com.ar/recibos/2022-07-01/mas-carro>

CANT.	DESCRIPCION	P (Unidades)	IMPORTE
3	Troncos de Leña N° 1 (1000g)	30	90.-
Pagos de Troncos - Cta. N° 22 x			
20 de Noviembre 2022			
con PAGO pendiente			
TOTAL		90.-	

CANT.	DESCRIPCION	P (Unidades)	IMPORTE
3	Troncos de Leña N° 1 (1000g)	30	90.-
Pagos de Troncos - Cta. N° 22 x			
20 de Noviembre 2022			
con PAGO pendiente			
TOTAL		90.-	

Notar que **no hay** una biyección entre URIs y recursos.

Es una **relación 1 a N**

- Un recurso puede ser apuntado por muchas URIs
- Pero ningún URI puede apuntar a más de un recurso

La World Wide Web: recursos y sus identificadores

Una **URI** toma la forma:

[esquema]:[estructura-dentro-del-esquema]

Ejemplos de URIs:

mailto:decano@exactas.uba.ar

https://comprafacil.com.ar/recibos/22874

ftp://bajaarchivos.net/musica/arjona.zip

urn://isbn:0451450523

urn://auto-arg-patente-AC040JJ

URLs

URNs

La World Wide Web: recursos y sus identificadores

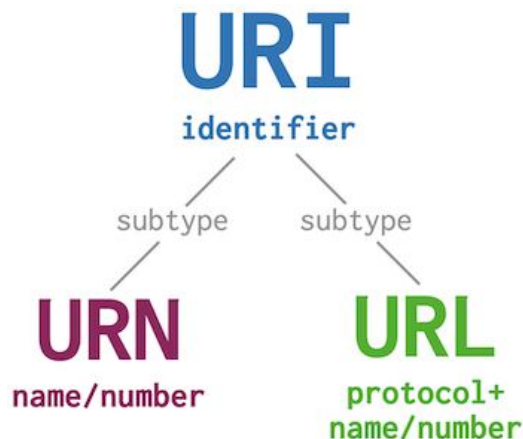
Una **URI** puede ser o bien una URL o una URN

- URL: **uniform resource locator**

Además de identificar un recurso, nos da información para encontrarlo

- URN: **uniform resource name**

Simplemente identifica un recurso mediante un nombre; no nos dice cómo encontrarlo



La World Wide Web: recursos y sus identificadores

Ejemplos de URLs:

mailto:decano@exactas.uba.ar

Usar protocolo SMTP para mandar un email a la dirección dada

https://comprafacil.com.ar/recibos/22874

Usar protocolo DNS para ubicar el servidor comprafacil.com.ar y luego enviar un request HTTP a recibos/22874

ftp://bajaarchivos.net/musica/arjona.zip

Usar protocolo DNS para ubicar el servidor bajaarchivos.net, luego hacer chdir a musica y pedir el archivo arjona.zip

La World Wide Web: recursos y sus representaciones

¿Cómo hacemos para leer o manipular un recurso?

<https://comprafacil.com.ar/recibos/22874>

[illegible]

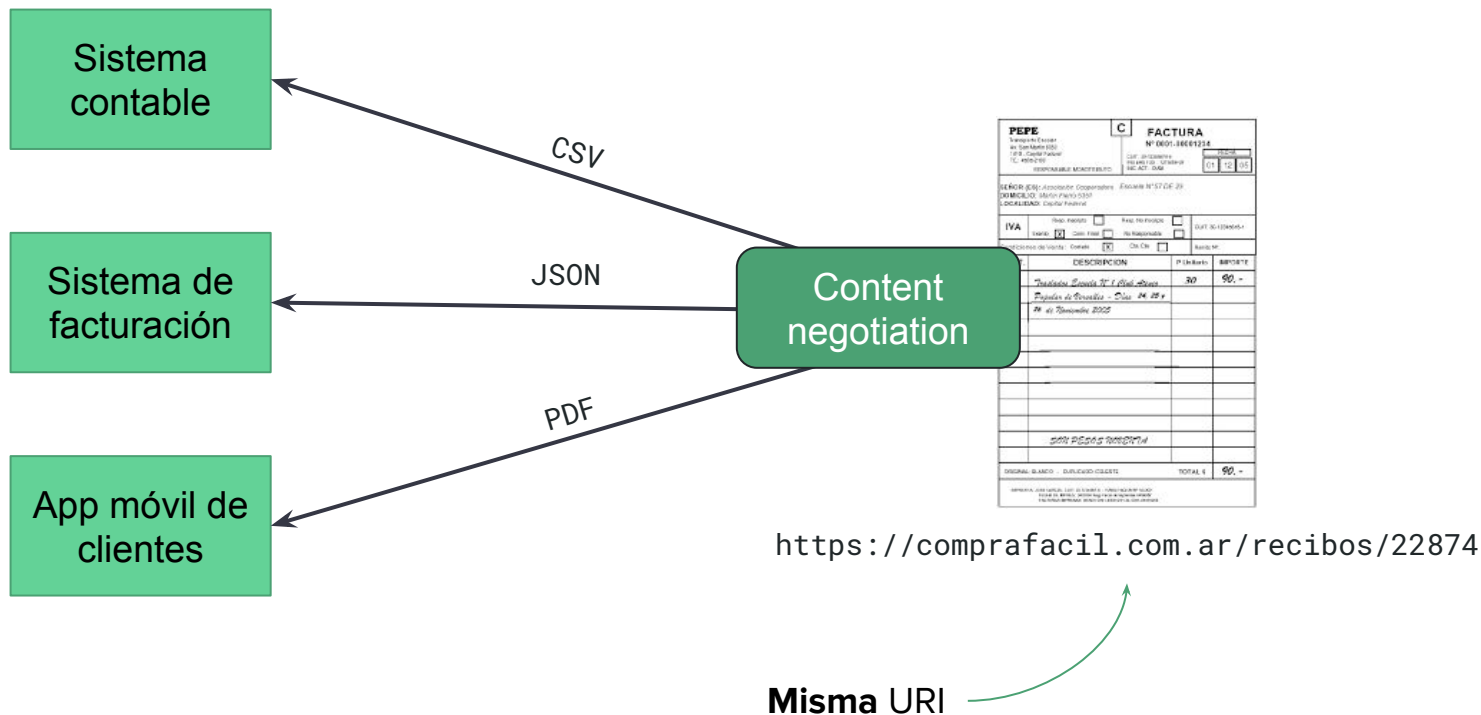
¿Nos interesa una captura JPG del recibo?

¿Un itemizado en forma de tabla CSV/Excel?

¿Un itemizado en formato JSON?

Un recurso suele tener múltiples **representaciones**.

La World Wide Web: recursos y sus representaciones



URIs/URLs/content negotiation - DEMO



<https://app.swaggerhub.com/apis/apple313/Petstore-Example/1.0.0>

La World Wide Web: verbos

Ya sabemos identificar y representar un recurso. Ahora, ¿qué queremos hacer con él?

HTTP nos permite utilizar una serie de **verbos** sobre un recurso:

- **GET:** obtener una representación del recurso
- **HEAD:** similar a GET pero sólo obtener los metadatos del recurso sin su representación
- **POST:** indicarle al recurso que procese un pedido
- **PUT:** modificar el estado del recurso
- **PATCH:** similar a PUT pero con una actualización de estado parcial
- **DELETE:** borrar un recurso
- **OPTIONS:** listar los verbos soportados por el recurso

También **TRACE/CONNECT** que no son frecuentemente usados y los dejaremos de lado

La World Wide Web: request

Para interactuar con un recurso a través de un verbo, lo que hacemos es un **request**

HTTP Request:

- **Verbo** a realizar. Por ej: GET
- **URI** del recurso sobre el que queremos trabajar. Por ej: <http://google.com.ar>
- Metadatos del pedido: cero o más **headers**. Por ejemplo para seleccionar la representación deseada mediante *content negotiation*
- Cuerpo o **body** del pedido: opcional, pero indispensable en requests tipo POST/PUT/PATCH

La World Wide Web: response

La respuesta a un request HTTP vendrá en forma de un **response**

HTTP Response:

- **Status code.** Código numérico que indica si el request se pudo cumplir
 - Ejemplo: 200 OK, 404 Not Found, 5xx Server Error
- Metadatos de la respuesta: cero o más **headers** por ejemplo indicar la representación usada
- Cuerpo o **body** de la respuesta.

La World Wide Web: status codes

Se pueden agrupar los status codes por centenas

- 100-199: el request está en curso
- 200-299: el request fue éxito
- 300-399: el recurso se movió (ej: redirect)
- 400-499: el request no es válido
- 500-599: el request es válido pero el server tuvo un problema al procesarlo (bug)

HTTP status ranges in a nutshell:

1xx: hold on

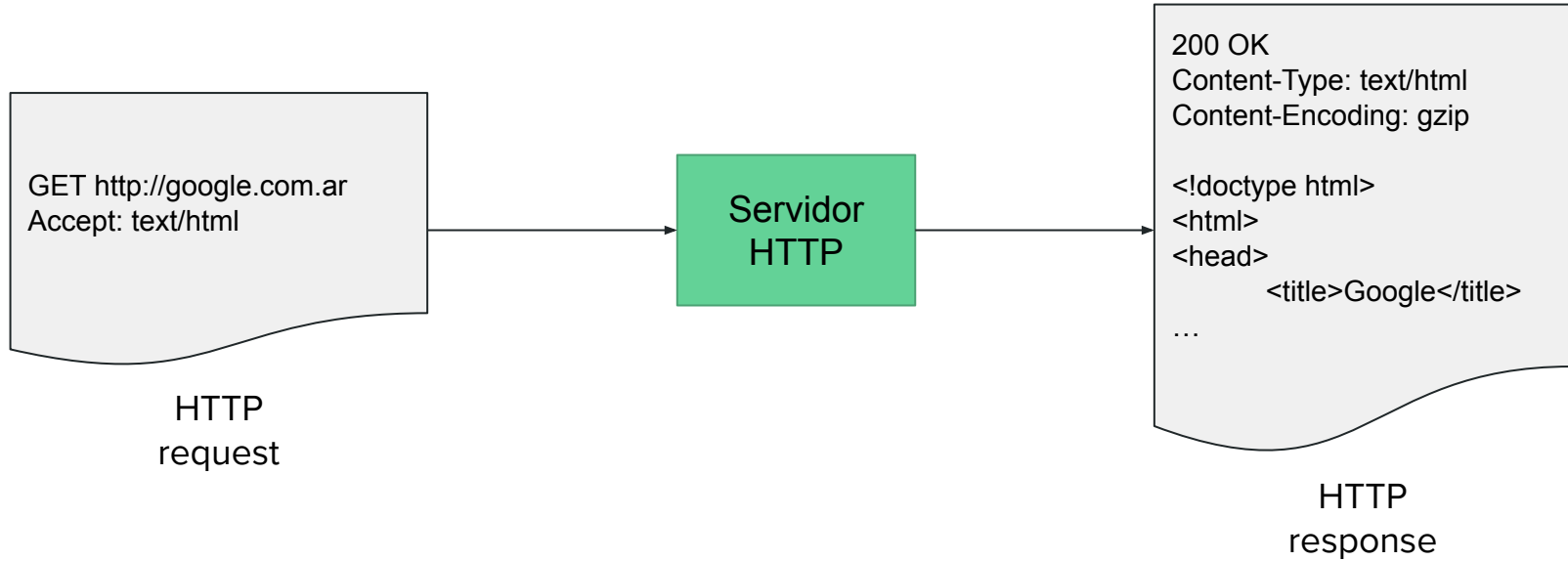
2xx: here you go

3xx: go away

4xx: you fucked up

5xx: I fucked up

La World Wide Web: ejemplo de request/response



Roy Fielding

La WWW tuvo un éxito fenomenal y los académicos querían entender el porqué.

Año 2000: Un estudiante de doctorado llamado Roy Fielding publica su tesis “Architectural Styles and the Design of Network-based Software Architectures”

- Provee una generalización del estilo arquitectónico de la WWW
- A este estilo lo llamó “Representational State Transfer” o **REST**



Roy Fielding y la definición de REST

Según Fielding:

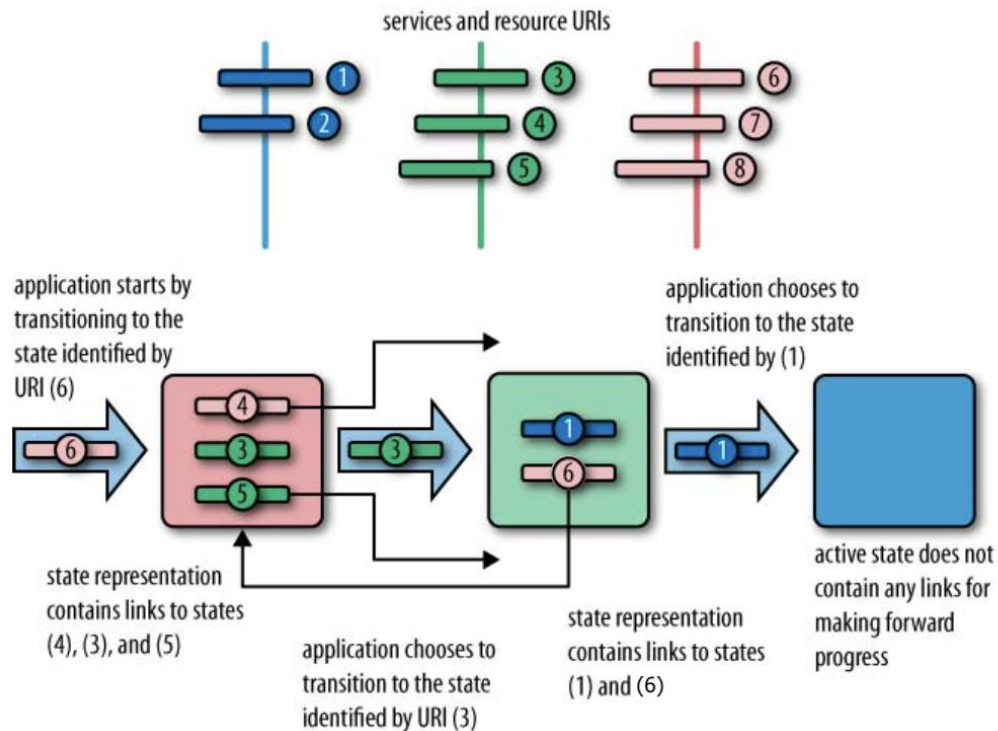
“La WWW es una aplicación hypermedia distribuida en la que sus recursos se comunican mediante el intercambio de representaciones de sus estados.”

Conceptos claves:

- Recursos con URIs
- Conjunto limitado de operaciones con semántica bien definida (verbos)
- Infraestructura ubicua



Ejecución en Representational State Transfer



Una máquina de estados

Pero descubrimos los posibles estados siguientes:

- Recién al avanzar (no de antemano)
- Y como parte de la respuesta en forma de **links**

A este modelo de ejecución se lo llama Hypermedia as the Engine of Application State (**HATEOAS**)

La Web como una plataforma de aplicaciones

La WWW surge como una plataforma para compartir **documentos multimedia**

Pero el estilo REST que gobierna la WWW **no nos limita** a otros tipos de aplicaciones

¿Qué hace a la web tan atractiva?

- **Soporte tecnológico**

Todos los sistemas operativos y lenguajes de programación soportan HTTP, JSON, etc.

- **Escalabilidad y performance**

No sería posible la escala de la web si no fuera por la enorme inversión en caches, servidores, etc. ¿Por qué no aprovechar esta inversión para desarrollar otras apps?

- **Bajo acoplamiento**

La web está diseñada para tolerar fallas (por ej. status 404) y no impone un gobierno centralizado, ni reglas de consistencia/integridad de datos.

Niveles de madurez de Richardson

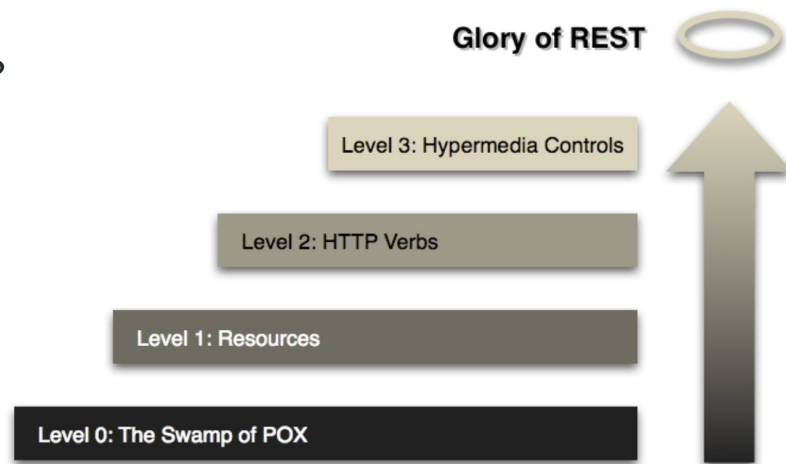
Usemos entonces la web para crear nuestras APIs

¿Pero qué tanto de los principios de la web usaremos?

- ¿Usamos URIs?
- ¿Usamos verbos?
- ¿Usamos hypermedia?

En 2008, Leonard Richardson planteó un modelo con 4 niveles de madurez

¿REST o “RESTful”?



Richardson - nivel 0

En el nivel 0 usamos la web sólo como transporte

Típicamente ya teníamos una API (por ej: RPC) y nos interesa exponerla vía web

- Un único recurso
- Un único verbo (típicamente GET/POST)
- No hay hypermedia

Ejemplos:

POST `http://mibilleteravirtual.com/api/v2`

Body:

```
{billetera: 22, operacion: "cargar saldo", monto: 200}
```

POST `http://mibilleteravirtual.com/api/v2`

Body:

```
{billetera: 22, operacion: "comprar", tienda: "coto", monto: 300}
```

Richardson - nivel 1

En el nivel 1 usamos **recursos**

- Varios recursos
- Un único verbo (típicamente GET/POST)
- No hay hypermedia

Ejemplos:

POST `http://mibilleteravirtual.com/billetera/22`

Body:

`{operacion: "cargar saldo", monto: 200}`

POST `http://mibilleteravirtual.com/billetera/23`

Body:

`{operacion: "comprar", tienda: "coto", monto: 300}`

Richardson - nivel 2

En el nivel 2 usamos recursos y también **verbos**

- Varios recursos
- Varios verbos
- No hay hypermedia

Ejemplos:

POST `http://mibilleteravirtual.com/billetera/22/recargas`

Body:

`{saldo: 200}`

DELETE `http://mibilleteravirtual.com/billetera/22`

GET `http://mibilleteravirtual.com/billetera/22`

Richardson - nivel 3

En el nivel 3 usamos recursos, verbos y hypermedia (HATEOAS)

Ejemplos:

GET <http://mibilleteravirtual.com/billetera/22>

Response: {

saldo: 1870.99,

links: {

“movimientos”: “GET /billetera/22/movimientos”,

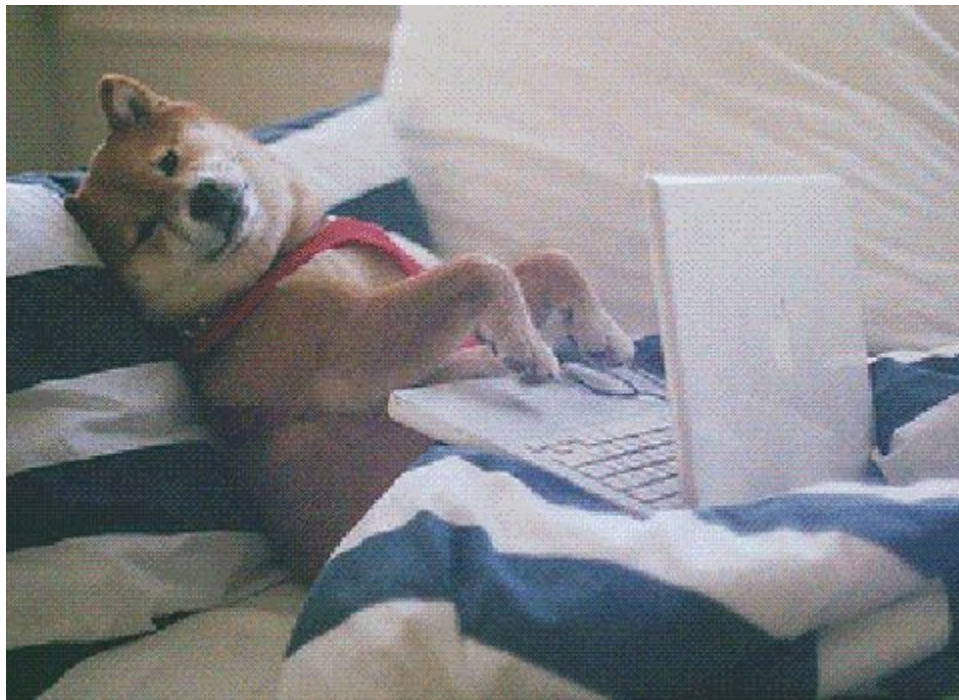
“desuscripción”: “DELETE /billetera/22”

}

}

DELETE <http://mibilleteravirtual.com/billetera/22>

Niveles de madurez de Richardson - DEMO



[Link](#)

Modelado de APIs REST

¿Cómo hacemos para descubrir los recursos de una API?

- HATEOAS nos ayuda en cada paso...
- ¿Pero por dónde empezamos? ¿Cómo hacemos el primer request?

¿Y qué hacemos si la API no está en el nivel 3 de Richardson?

- ¿Cómo sabemos qué recursos y que verbos se pueden usar?

Los contratos al rescate

- 2011. **Swagger**. Lenguaje para modelar APIs REST
- 2013. **API Blueprint** y **RAML**, otras variantes para modelar APIs REST
- 2016. Swagger pasa a llamarse **OpenAPI Spec (OAS)** y es el principal standard hoy en día
 - Hoy se utiliza principalmente OAS v3, también llamado OAS3

OpenAPI Spec v3

```
title: Sample Pet Store App
summary: A pet store manager.
description: This is a sample server for a pet store.
termsOfService: https://example.com/terms/
contact:
  name: API Support
  url: https://www.example.com/support
  email: support@example.com
license:
  name: Apache 2.0
  url: https://www.apache.org/licenses/LICENSE-2.0.html
version: 1.0.1
servers: ...
paths: ...
components: ...
```


OpenAPI Spec v3 - servers

```
servers:  
- url: https://development.gigantic-server.com/v1  
  description: Development server  
- url: https://staging.gigantic-server.com/v1  
  description: Staging server  
- url: https://api.gigantic-server.com/v1  
  description: Production server
```

OpenAPI Spec v3 - paths

```
/pet/{petId}:  
  delete:  
    description: Deletes a pet  
    parameters:  
      - name: petId  
        in: path  
        description: Pet id to delete  
        required: true  
        schema:  
          type: integer  
          format: int64  
    responses:  
      '400':  
        description: Invalid ID supplied  
      '404':  
        description: Pet not found
```

OpenAPI Spec v3 - paths

```
/pets:
  get:
    description: Returns all pets from the system that the user has access
to
    responses:
      '200':
        description: A list of pets.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/pet'
```

OpenAPI Spec v3 - components

```
components:
  schemas:
    Pet:
      type: object
      required: [id, status]
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
          example: doggie
        status:
          type: string
          description: pet status in the store
          enum: [available, pending, adopted]
```

OAS3 - DEMO



<https://app.swaggerhub.com/apis/apple313/Petstore-Example/1.0.0>

Buenas prácticas en APIs REST - URIs I

Usamos “/” para denotar jerarquía en una URI

```
https://supermercado.com.ar/gondolas/verdulería/frutas/kiwi
```

La jerarquía de la URL debería ser con recursos “en escalera”, todos los siguiente son recursos:

```
https://supermercado.com.ar/gondolas/verdulería/frutas
```

```
https://supermercado.com.ar/gondolas/verdulería
```

```
https://supermercado.com.ar/gondolas
```

Usamos “-” si tenemos palabras compuestas o con espacios en nuestras URIs

```
https://supermercado.com.ar/gondolas/quesos-y-fiambres
```

Buenas prácticas en APIs REST - URIs II

No usamos extensiones de archivos en nuestras URIs (para eso está *Content Negotiation*)

`https://supermercado.com.ar/pedidos/9838/recibo.json` ❌

Nunca usamos "/" al final de una URI

`https://supermercado.com.ar/gondolas/` ❌

Nunca usamos mayúsculas en una URI

`https://supermercado.com.ar/gondolas/Limpieza` ❌

Si bien es legal, tratar de evitar usar símbolos como tildes o ñ en las URIs

`https://supermercado.com.ar/gondolas/limpieza/paños` ⚠️

Tipos de recursos en una API REST

Hay categorías de recursos que son recurrentes, surgen los siguientes patrones de recursos:

- **Documentos**

- Contienen información sobre un elemento específico
- Ej: `https://supermercado.com.ar/gondolas/verduleria/frutas/kiwi`
- Ej: `https://supermercado.com.ar/gondolas/verduleria`

- **Colecciones**

- Representan un conjunto de elementos
- Ej: `https://supermercado.com.ar/gondolas/verduleria/frutas`
- Ej: `https://supermercado.com.ar/gondolas`

- **Controladores**

- Representan un proceso de negocio que se puede disparar (más de esto la clase que viene)
- Ej: `https://supermercado.com.ar/pedidos/9838/facturar`

Buenas prácticas en APIs REST - tipos de recursos

Las URIs de recursos de tipo **documento** usan el singular

`https://supermercado.com.ar/repartidores/juan`



Las URIs de recursos de tipo **colección** usan el plural

`https://supermercado.com.ar/gondolas`

Las URIS de recursos de tipo **controlador** usan verbos

`https://supermercado.com.ar/repartidores/juan/notificar`

Las operaciones de tipo ABM (alta, baja, modificación) no deben figurar en las URIS

`https://supermercado.com.ar/repartidores/crear` 
`https://supermercado.com.ar/pedidos/9838/borrar` 

Query parameters

Del RFC 3986 que define las URIs:

URI = scheme "://" authority "/" path ["?" **query**] ["#" fragment]

La parte de la query se corresponde a pares `param=value` donde el value es opcional

Puede suceder que un parámetro figure múltiples veces

Ejemplos:

`?ordenarPor=nombre&criterio=descendente`

`?ordenarPor=nombre&ordenarPor=apellido&ordenarPor=edad`

`?ordenarPor=edad&eliminarDuplicados`

`?eliminarDuplicados=true`

Buenas prácticas en APIs REST - query parameters

Los query parameters se colocan en *camelCase*

`https://supermercado.com.ar/gondolas?ordenarPor=nombre`

Podemos usar query parameters para filtrar colecciones

`https://supermercado.com.ar/gondolas/verdulería/frutas?precioMax=400`

Podemos usar query parameters para paginar colecciones

`https://supermercado.com.ar/pedidos?offset=40&limit=10`

Buenas prácticas en APIs REST - verbos I

No abusar del verbo POST sólo porque un cliente podría no conocer cómo usar otros verbos

POST `https://supermercado.com.ar/pedidos/9938/cancelar` ❌

DELETE `https://supermercado.com.ar/pedidos/9938` ✅

El verbo PUT se utiliza para update o insert (“upsert”) de documentos

PUT `https://supermercado.com.ar/clientes/223/medio-de-pago-favorito`
{ tipo: “tarjeta-de-credito”, detalle: “2387-2772-8776-8444” }

El verbo PUT se utiliza para actualizar recursos mutables

PUT `https://supermercado.com.ar/clientes/223`
{ nombre: “Juana”, apellido: “Barrientos” }

Buenas prácticas en APIs REST - verbos II

Usamos POST para agregar elementos a una colección

POST `https://supermercado.com.ar/gondolas/verduleria/frutas`
`{ id: "papaya", precio: 233.52 }`

Usamos DELETE para quitar elementos de una colección

DELETE `https://supermercado.com.ar/gondolas/verduleria/frutas/papaya`

Usamos POST para disparar controladores


POST `https://supermercado.com.ar/pedidos/9838/facturar`
`{ direccionDeFacturacion: "Rivadavia 2256 4A, CABA" }`

Buenas prácticas en APIs REST - status codes I

El status 200 OK se usa para indicar éxito

```
GET https://supermercado.com.ar/gondolas/verduleria/frutas  
> 200 OK
```

Nunca usamos 200 OK si hay un error en el cuerpo de la respuesta

```
GET https://supermercado.com.ar/gondolas/limpieza/frutas  
> 200 OK   
> { error: "Categoría frutas no es válida dentro de góndola limpieza" }
```

Buenas prácticas en APIs REST - status codes II

El status 201 CREATED se usa para indicar que se creó un recurso de manera sincrónica

```
POST https://supermercado.com.ar/gondolas/verduleria/frutas
{ nombre: "papaya", precio: 233.52 }
> 201 CREATED
```

El status 202 ACCEPTED se usa para indicar que se inició algún proceso de forma asincrónica

```
POST https://supermercado.com.ar/pedidos/9838/facturar
{ direccionDeFacturacion: "Rivadavia 2256 4A, CABA" }
> 202 ACCEPTED
```

El status 204 NO CONTENT se usa para indicar que se borró un recurso

```
DELETE https://supermercado.com.ar/gondolas/verduleria/frutas/papaya
> 204 NO CONTENT
```

Buenas prácticas en APIs REST - status codes III

El status 400 BAD REQUEST se usa para indicar algún problema general con el pedido

```
POST https://supermercado.com.ar/gondolas?offse=10
> 400 BAD REQUEST
> { error: "query param 'offse' no es reconocido" }
```

El status 401 UNAUTHORIZED se usa para indicar que hubo un problema con las credenciales

```
GET https://supermercado.com.ar/mi-cuenta?apiToken=93738
> 401 UNAUTHORIZED
> { error: "El token provisto caducó" }
```

El status 403 FORBIDDEN se usa para indicar que el request no es realizable por un tema de permisos cuando las credenciales son correctas (sino sería un 401).

```
DELETE https://supermercado.com.ar/gondolas/verduleria
> 403 FORBIDDEN
```


Buenas prácticas en APIs REST - status codes IV

El status 404 NOT FOUND se usa cuando la URI provista no apunta a ningún recurso existente

```
POST https://supermercado.com.ar/gondolas/vestimenta  
> 404 NOT FOUND
```

El status 405 METHOD NOT ALLOWED se usa para indicar que un verbo no es válido en una URI

```
DELETE https://supermercado.com.ar  
> 405 METHOD NOT ALLOWED
```

El status 500 INTERNAL SERVER ERROR se usa para indicar que el request era válido, pero algo pasó durante el procesamiento del mismo; típicamente un bug. Dar más información o no sobre qué pasó es un tradeoff entre mantenibilidad/seguridad

```
GET https://supermercado.com.ar  
> 500 INTERNAL SERVER ERROR  
> { "error": "el servidor falló en la línea 7 del archivo gondolas.py" }
```