

API Rest

Día 2

REST Web Services

REST (Representational State Transfer), es un estilo de arquitectura de software dirigido a sistemas hipermedias distribuidos como lo es la web.

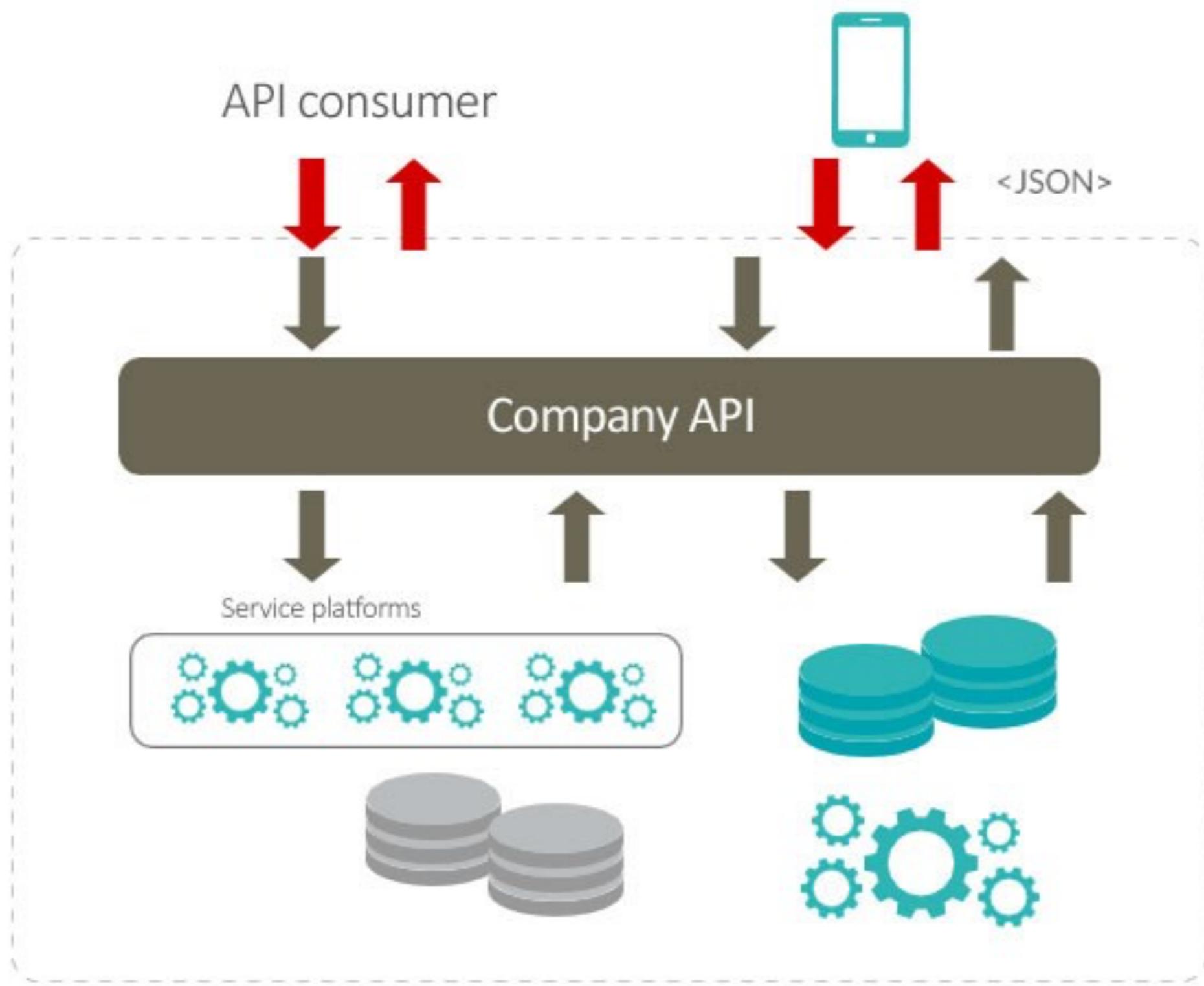
Este término se refiere específicamente a una colección de principios para el diseño de arquitecturas en red.

Existen varios proyectos que pueden verse beneficiados de una arquitectura REST. Concretamente aquellos en los que la idea principal está en la manera en la que se hacen las peticiones al servidor desde el cliente, basados en el recurso de interés.

REST Web Services

Es importante que la arquitectura REST cumpla con 6 principios:

- Cliente – Servidor
- Interface Uniforme
- Capaces de almacenar en caché
- No manejan estado
- Sistemas en capas
- Código baja demanda



Descripción General

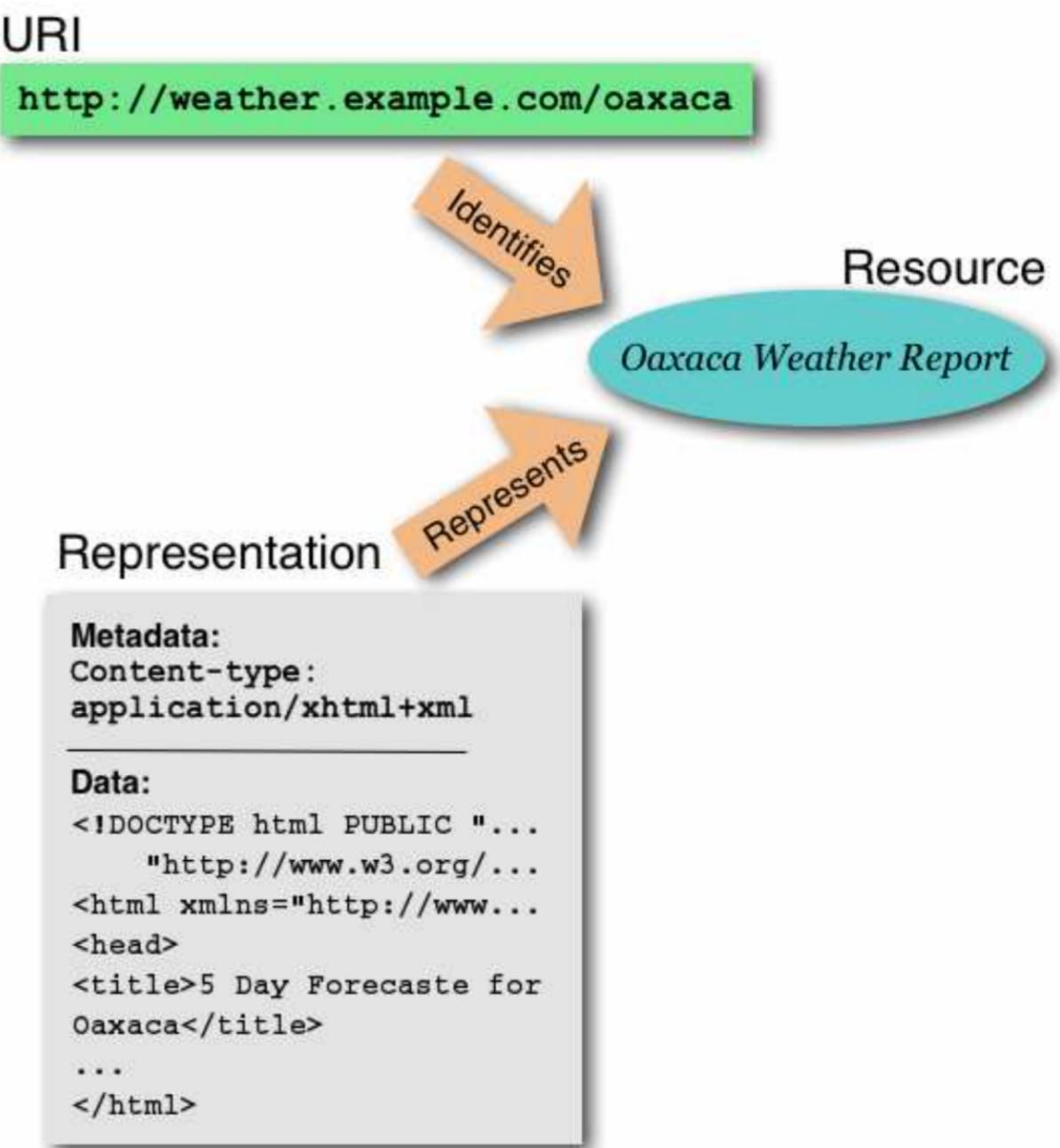
- Por qué surge REST
 - WS-* es muy completo pero muy complejo
- Es un estilo de arquitectura, no es una especificación ni un estándar
- Surge de la tesis de doctorado de Roy Fielding
 - Define restricciones para que un sistema sea REST
 - Se basa en la Web estática

Descripción General

- Principios o Restricciones
 - Recursos deben ser uniformemente accesibles (URI única)
 - Recursos son accedidos y actualizados por operaciones GET, POST, PUT, DELETE
 - Metadatos para describir recursos
 - Comunicación entre cliente y servidor debe ser *stateless*

Descripción General

- Recursos
 - Cualquier cosa que pueda identificar usando una URI
- URI
 - Dirección universal a un recurso
- Representación
 - Lo que se envía al cliente cuando solicita un recurso



Descripción General

- Funcionamiento
 - Orientado a recursos, no a operaciones
 - Una invocación REST es un *request* HTTP
 - Las operaciones sobre recursos son limitadas

Operación	Método HTTP
Create	POST
Retrieve	GET
Update	PUT
Delete	DELETE

Descripción General

- Ejemplo de invocación REST

```
Encabezados de la petición
GET /sqlrest/INVOICE/9999/ HTTP/1.1
Host: www.thomas-bayer.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; es-AR; rv:1.9.2.3) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es,en;q=0.8,es-ar;q=0.5,en-us;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive

Método
```

URI de la solicitud

```
Encabezados de la respuesta
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Wed, 07 Apr 2010 01:30:00 GMT

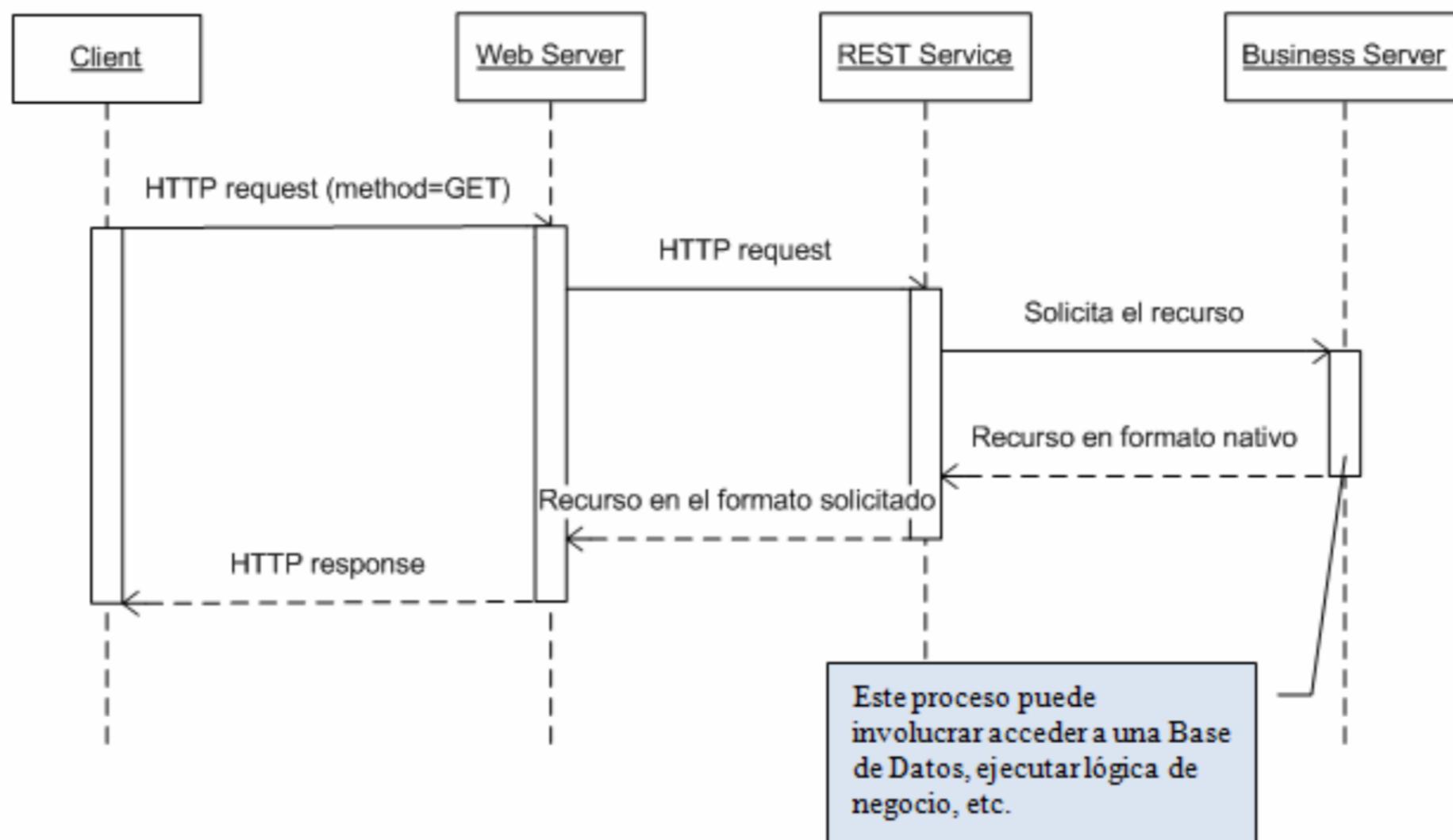
Código de respuesta
```

Formatos aceptados para la representación

Formato en que se envía la respuesta

Descripción General

- Que ocurrió en el servidor



Descripción General

- Servicios REST híbridos
 - Servicios que dicen ser REST pero no lo son totalmente
 - Ejemplo: servicio Flickr de Yahoo

Llamada RPC con
apariencia REST

```
http://api.flickr.com/services/rest?method=flickr.g  
alleries.getList&api_key=1111
```

```
http://api.flickr.com/services/rest?method=flickr.g  
alleries.create&api_key=1111
```

Se crean recursos
usando método GET

Descripción General

• REST vs WS-*

REST	WS-*
Orientado a recursos	Orientado a operaciones
Busca escalabilidad y performance	Busca interoperabilidad
Ve la Web como un gran repositorio	Ve la Web como medio para intercambio de mensajes
Conjunto limitado de operaciones	Operaciones ilimitadas
Servidor <i>stateless</i>	<i>Stateless</i> o <i>stateful</i>
Navegación a través de links	Grafo de estados debe conocerse
No hay estándares de seguridad, transaccionalidad, etc.	Existe stack WS-*

Arquitectura orientada a recursos

- Es importante formalizar una arquitectura
- Basado en axiomas
 - Los recursos son abstractos y se pueden representar por al menos una URI
 - Las representaciones son inmutables
 - Existe al menos una URI para cada recurso
 - Cada solicitud al servidor es independiente de todas las otras

Arquitectura orientada a recursos

- Navegabilidad
 - Brindada a través de links en las propias representaciones
- Interface única
 - Usar métodos GET, POST, PUT, DELETE
 - No usar GET para crear recursos
 - En lo posible usar operaciones idempotentes

Arquitectura orientada a recursos

- Guía para el diseño de un servicio REST
 - Identificar los recursos del Sistema
 - Diseñar las URLs
 - Definir las operaciones disponibles sobre cada recurso
 - Definir tipos de representación que se aceptarán
 - Definir un esquema de seguridad

Laboratorios API Rest

Gracias

