REST

REpresentational State Transfer

*Ignacio Muñoz Vicente @imunoz_*

¿Qué es REST?

- Es una **técnica de arquitectura software** para sistemas hipermedia distribuidos como la World Wide Web

Wikipedia

¿Qué no es REST?

- REST **no es un servicio web ni una tecnología**

¿Qué son servicios RESTful?

- Son aquellos servicios web **que cumplen los principios de REST**

Características

- **Sin estado** - Cada mensaje HTTP contiene toda la información necesaria
- **Operaciones HTTP** - Uso de POST, GET, PUT, DELETE y otros
- **Sintaxis universal: URI** - Cada recurso queda identificado por su URI
- **Hipermedios** - Relación entre recursos mediante hipervínculos
- **Códigos de estado HTTP** - Usa los códigos HTTP para indicar el resultado de la petición

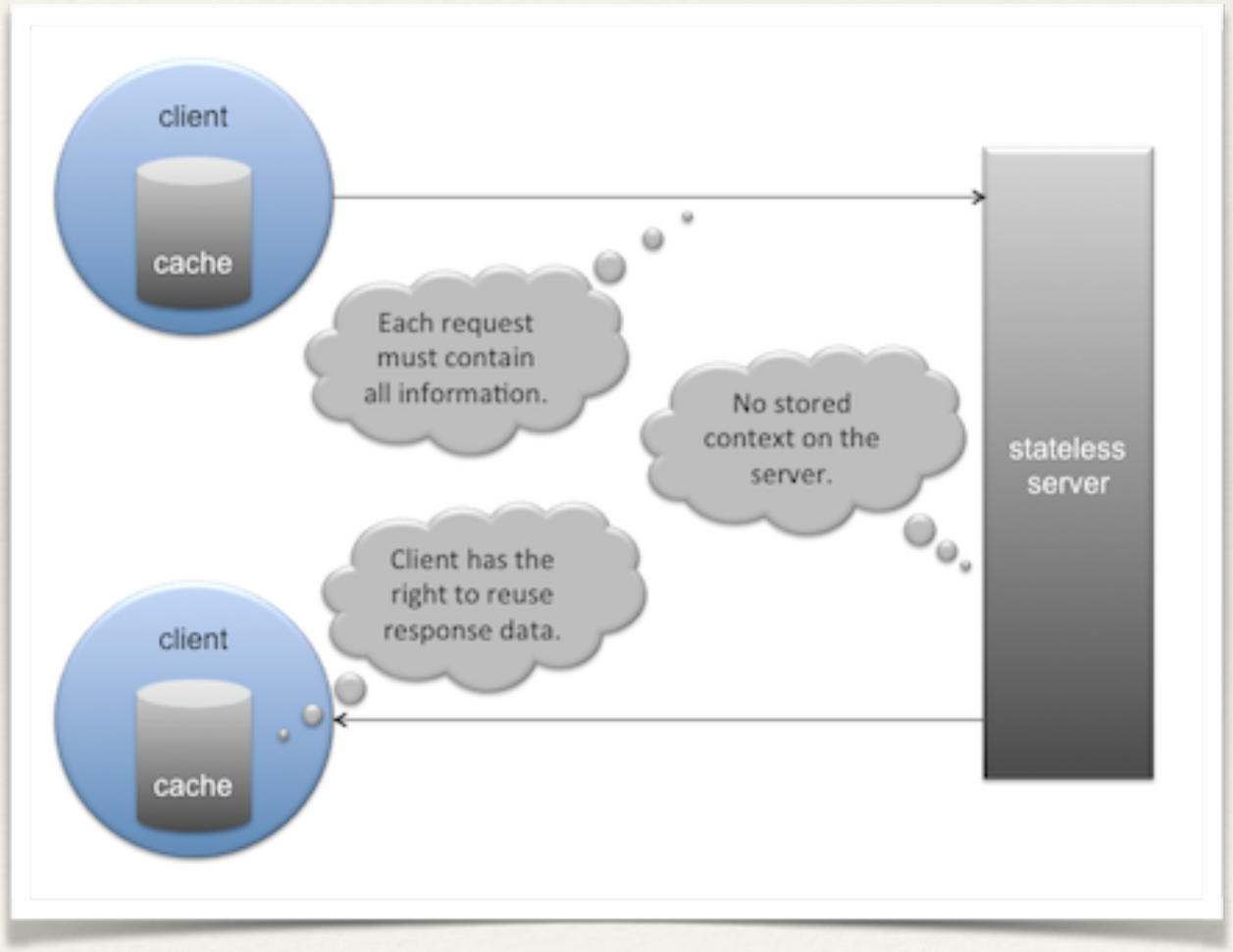
Características

Sin estado

- Sin sesión ni cookies

- Cada petición es independiente

- Cada petición debe tener toda la información necesaria para que el servidor la entienda y devuelva la respuesta correcta



Características

Operaciones HTTP

Utiliza los métodos HTTP de forma explicita para las operaciones a realizar:

- **GET**: obtener un recurso del servidor

- **POST**: Crear un recurso en el servidor

- **PUT**: Actualizar un recurso en el servidor

- **DELETE**: Eliminar un recurso del servidor

- Hay más métodos pero estos son los más frecuentes: **CRUD**

Características

Operaciones HTTP

Utiliza los métodos HTTP de forma explicita para las operaciones a realizar:

- **GET**: obtener un recurso del servidor

- **POST**: Crear un recurso en el servidor

- **PUT**: Actualizar un recurso en el servidor

- **DELETE**: Eliminar un recurso del servidor

- Hay más métodos pero estos son los más frecuentes: **CRUD**

GET /usuarios Nos permite acceder al listado de usuarios.

POST /usuarios Nos permite crear un usuario nuevo. Se envía en la petición la información que se necesite para la creación (mediante JSON o XML):

```
<usuario>
    <nombre>Ignacio</nombre>
    <apellido>Muñoz</apellido>
</usuario>
```

GET /usuarios/12 Nos permite acceder al detalle del usuario con id 12.

PUT /usuarios/12 Permite editar el usuario con id 12. Al igual que en POST se pasan los datos necesarios como cuerpo de la petición:

```
<usuario>
    <apellido>Vicente</apellido>
</usuario>
```

DELETE /usuarios/12 Eliminar el usuario con id 12.

Características

Sintaxis universal

- Cada recurso se identifica con su URI única

- Nombres mejor que verbos (excepto cuando son operaciones)

- Nombres mejor en plural: *http://_____/usuarios/*

- Normalmente dos URLs similares: para listado (collection) y para ítem

- Versión de la API en la URL

- CRUD a través de su URI y una operación HTTP

Características

Hipermedia

- HATEOAS: *Hypermedia As The Engine Of Application State*

- Una respuesta debe ofrecer toda la información necesaria

- Dado un punto de entrada a la API REST, es posible descubrir todos sus recursos a partir sólo de las respuestas del servidor .La respuesta devuelta por el servidor puede contener hipervínculos a otros recursos

```
{
   "id": 12,
   "nombre": "Ignacio",
   "apellido": "Muñoz",
   "dispositivos": [
      { "id": 1033 },
      {"id": 3889 }
   ]
}
```

```
{
   "id": 12,
   "nombre": "Ignacio",
   "apellido": "Muñoz",
   "dispositivos": [
      { "dispositivo": http://…/usuarios/12/dispositivos/1033 },
      {"dispositivo": http://…/usuarios/12/dispositivos/3889 }
   ]
}
```

Características

Códigos de estado

- Indican el resultado de la operación

- Hay que tener cuidado con:

  401 "Unauthorized"
  significa realmente *"Unauthenticated"*

  403 "Forbidden"
  significa realmente *"Unauthorized"*

- Si se quiere dar más info de un error:
  - Errores internos (complementan al código de error de HTTP)
  - Descripción

ALGUNOS CÓDIGOS

200 - Petición correcta.

201 - Petición completada y recurso creado correctamente.

202 - Petición aceptada pero no completada.

304 - No modificado.

400 - Solicitud incorrecta.

401 - No autorizado.

403 – Prohibido.

404 - No encontrado.

405 - Método no permitido.

406 - No aceptable. El servidor no puede devolver la información en el formato solicitado en la petición.

500 - Error interno del servidor.

501 - No implementado.

Características

MediaTypes

- En la arquitectura REST **no existe un formato de intercambio** de información predefinido

- Se indica en las cabeceras / headers:

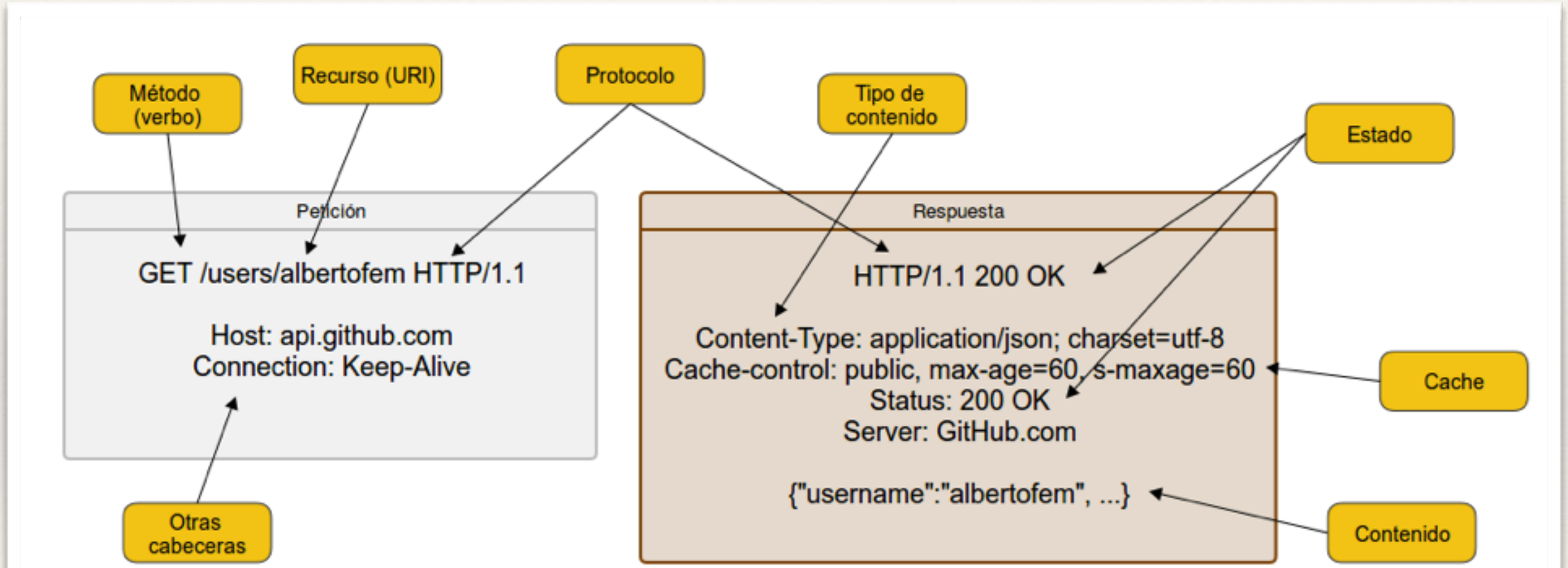
    - Header ***Accept*** en la petición

    - Header ***Content-Type*** en la respuesta

- Ejemplo: *application/json*

- Valores múltiples: *Accept: application/json, text/plain*

Ejemplo petición



Recomendaciones

- Número de versión de la API en la URL
  - *http://<URL>/v3/usuarios*
- Unificar JSON y JavaScript
  - *camelCase* (así nos ahorraremos 'mapear' luego el JSON)
- Paginación con 2 valores: offset + limit
  - *http://<URL completa>?offset=X&limit=Y*
- Seguridad por recurso, no por URL:
  - OAuth2
  - Basic (SSL) Auth
  - hmac

Recomendaciones

- Si son acciones que no requieren recursos (por ejemplo hacer un cálculo numérico, mejor utilizar verbos en el nombre)
  - *http://<URL>/sumar?valor1=X&valor2=Y*
- Búsqueda: método search, parámetro q (URL encoded)
  - *http://<URL completa>?q=cualquier%20palabra*
- Para conocer métodos HTTP disponibles
  - Usar OPTIONS —> Allow header: HEAD, GET, PUT, POST, DELETE, OPTIONS, …
- Fechas mediante ISO 8601 (UTC)
  - *año-mes-día*T*hora:minutos:segundos,nanosegundos*Z
- Versión también en los datos
  - *Accept: application/vnd.github.v3+json*

Cacheo

Cacheo en servidor

- Hay que evitar a toda costa accesos a base de datos —> muy costosos

- Las peticiones de recursos más solicitadas deben cachearse en memoria

- Muchas soluciones:

    - **REDIS**: cache en memoria, clave-valor. *http://redis.io/*

    - **Varnish**: *https://www.varnish-cache.org/*

    - Otros muchos

Cacheo

Cacheo en cliente

El campo ETag indica la versión del recurso:

1- Cliente solicita un recurso

2- El servidor devuelve:

- Código 200
- Body: el recurso solicitado
- Header: cabeceras habituales y el ETag: *ETag: "64233457696a7c876b7e"*

3- Cliente vuelve a solicitar el recurso, enviado la siguiente cabecera:

If-None-Match: "64233457696a7c876b7e"

Si el recurso ha cambiado el servidor devolverá el recurso actualizado, nuevo ETag y código 200.

Si el recurso no ha cambiado el servidor devolverá:

- Código 304
- Body: vacío
- Header: cabeceras habituales y el mismo ETag

¿Por qué usar REST?

- Es sencillo de entender e implementar
- Escalabilidad, independencia, seguridad, encapsulación, etc.
- Mola, es lo más parecido a un estándar

¿Por qué usar REST?

- Es sencillo de entender e implementar
- Escalabilidad, independencia, seguridad, encapsulación, etc.
- Mola, es lo más parecido a un estándar

¿Cuando NO usar REST?

- Cuando es un servicio muy personalizado: seguridad, transacciones, vocabulario y nomenclatura específica, etc.
- Necesidad de respuesta asíncrona en la petición

REST

REpresentational State Transfer

*Ignacio Muñoz Vicente @imunoz_*

Fuentes:

http://es.wikipedia.org/wiki/Representational_State_Transfer
http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-ii-ejemplos
http://rest.elkstein.org/
http://haciendo.minube.com/usar-etags-109