Introducción a REST

Juan Quemada, DIT - UPM

©

Que es REST

- **REST: RE**presentational **S**tate **T**ransfer
  - El estado se representa en el recurso transferido al cliente
    - http://en.wikipedia.org/wiki/Representational_state_transfer

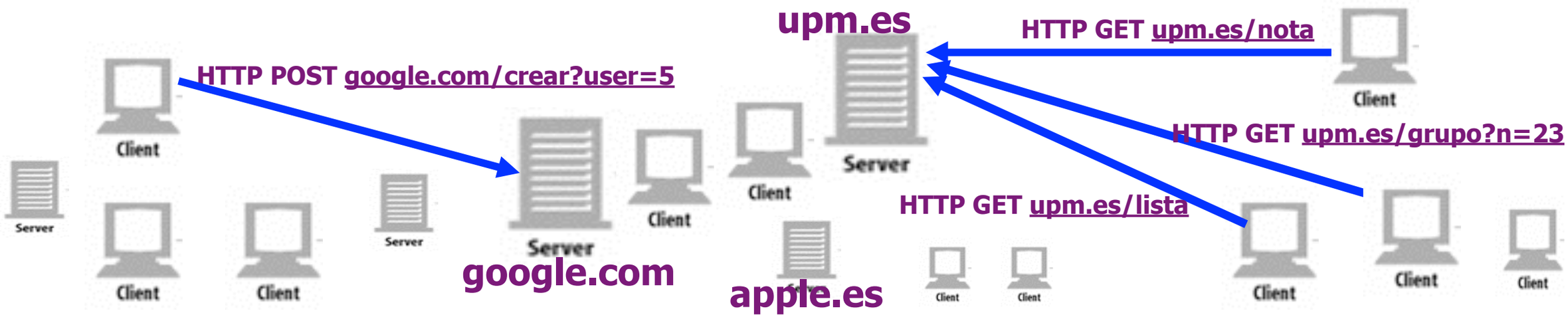
- **REST:** Principios arquitecturales para aplicaciones Web escalables
  - Propuestos por **Roy Fielding** en su Tesis Doctoral (2000)
    - Roy Fielding fue co-diseñador de HTTP y ha sido uno de los desarrolladores principales del proyecto Apache
  - Conocida como: **Arquitectura Orientada a Recursos (ROA)**

- Interfaces **REST** o Servicios Web **RESTful**
  - Cliente y servidor interaccionan con Interfaz Uniforme de HTTP
    - Métodos GET, POST, PUT y DELETE
  - REST está muy extendido: Google, Twitter, Amazon, Facebook, …

©

Interfaces REST

- Interfaz REST
  - Cliente y servidor **interaccionan con HTTP**
    - Cada operación identifica el recurso con una **ruta (path) diferente**
      - Por ejemplo, **/nota/5, /user/10, /notas/user/5, /grupo?n=23, ...**
- Solo utilizan métodos o comandos del **interfaz uniforme**
  - **GET:** trae al cliente (lee) un recurso identificado por un URL
  - **POST:** crea un recurso identificado por un URL
  - **PUT:** actualiza un recurso identificado por un URL
  - **DELETE:** borra un recurso identificado por un URL
  - ..... (HTTP tiene mas métodos, pero no pertenecen al interfaz uniforme)



Principios REST

- **Direccionabilidad** (Addressability) de los recursos
- Uso del **interfaz uniforme de HTTP:** GET, POST, PUT y DELETE
- **Comunicación sin estado** en el servidor (Statelessness)
- Servicio **hipermedia** (Connectedness) conectado con URLs
- **Recursos** en **formatos abiertos**: HTML, XML, JSON, RSS, texto plano, …

©

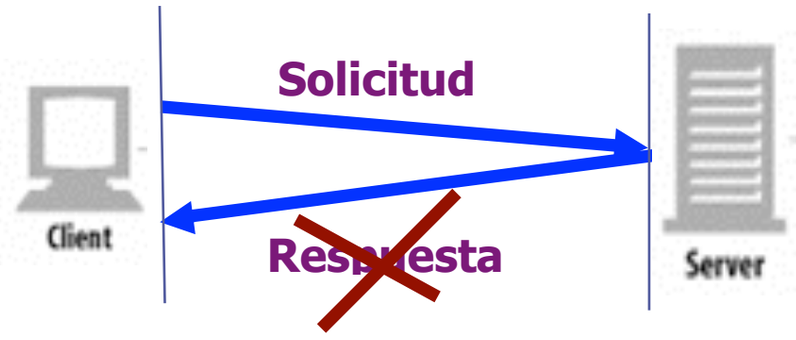
Direccionabilidad e interfaz uniforme

- En una arquitectura orientada a recursos
  - Todo recurso del servidor tiene una ruta (dirección) diferente
    - Las rutas (paths) son las direcciones (pueden incluir query o ancla para información adicional)
  - Los recursos se procesan solo con los métodos del interfaz uniforme
    - GET, POST, PUT y DELETE

- Ejemplo de una colección de usuarios (suele asociarse a una tabla de la DB)
  - La ruta suele tomar el nombre de la colección en plural (usuarios)
    - Las operaciones individuales identifican al usuario con un identificador en la ruta
      - **POST** **/usuarios?nombre=Pedro+Ramirez&edad=8** // Crear nuevo usuario
      - **GET** **/usuarios** // Traer lista de todos los usuarios
      - **GET** **/usuarios/2007** // Traer datos del usuario 2007
      - **DELETE** **/usuarios/2007** // Borrar usuario 2007 de la colección
      - **PUT** **/usuarios/2007?edad=9** // Actualizar edad del usuario 2007
      - ademas suele haber primitivas GET para cargar formularios asociados a POST y PUT

©

Seguridad e idempotencia



- Seguridad e idempotencia son 2 propiedades importantes
  - Método **seguro** (safe): no modifica datos en el servidor y puede ser cacheado
  - Método **idempotente:** el resultado es independiente del número de invocaciones

- **Operación idempotente**
  - El resultado de invocar el método n veces es igual a invocarlo 1 vez
    - Por ejemplo: **x=2** es idempotente, pero **x=x+1** no es idempotente
  - Las operaciones asociadas a un interfaz REST deben ser idempotentes
    - Por ejemplo: **PUT /usuario/2007?edad=9**

- **Internet no es fiable**
  - La invocación de una solicitud HTTP puede ejecutarse **n** veces en el servidor
    - Si la **solicitud se pierde** y hay reenvío, el método **se ejecuta solo 1 vez**
    - Si la **respuesta se pierde** y hay reenvío, el método **se ejecuta 2 veces** en el servidor

Propiedades del interfaz uniforme

- **Interfaz uniforme o CRUD**
  - Permite crear servicios desacoplados y escalables (http://restcookbook.com)
- **Propiedades** de los métodos del interfaz uniforme
  - POST: **El más peligroso** (puede duplicar recursos)
  - GET: **Seguro (cacheable) e idempotente**
  - PUT: **idempotente**
  - DELETE: **idempotente**
- Recomendaciones de diseño importantes
  - Tratar de minimizar el impacto de no idempotencia de **POST**
  - No utilizar **nunca GET** para **modificar recursos** del servidor
    - Utilizar POST, PUT o DELETE según el tipo de modificación, porque GET puede ser cacheado y no modificará los recursos

Servicio hipermedia sin estado en el servidor

- El **servicio se usa navegando** por los recursos recibidos del servidor
  - El recurso contiene el estado del cliente (suele ser una página Web)
    - Las **transiciones** son los enlaces (URLs) y se navega al hacer clic en ellos

- Los servidores escalan porque **no guardan el estado** de los clientes
  - Solo gestionan recursos a través del interfaz uniforme
    - Con transacciones HTTP que son independientes entre sí

- Los **clientes guardan siempre el estado de uso del servicio**
  - en la página Web o recurso cargadas, en cookies, en localStorage, ….

©

Representación de los recursos

- Los recursos transferidos por el servidor al cliente
  - Representan el estado y las transiciones necesarios para navegar por el servicio
    - Un mismo recurso se puede representar (o serializar) en distintos formatos

- Formatos más habituales de representación de recursos:
  - **HTML**: para presentar información legible en un browser
    - XHTML: versión sintacticamente más estricta de HTML
  - **JSON**: Formato de serialización de objetos Javascript
  - XML: Formato de datos tipo SGML del W3C
  - RSS: formato para representar colecciones (de feeds de blogs)
  - ATOM: formato para representar colecciones (de feeds de blogs)

- HTTP usa el tipo MIME para tipar los recursos que transfiere

©