actibyti] a netmind social project

SERVICIOS WEB RESTFUL

Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

ÍNDICE DE CONTENIDOS

1 CASO PRÁCTICO

Caso Práctico: Banana GEST RESTful

"BananaTube" es el proyecto estrella de Banana Apps.

BananaTube será el próximo boom! de las redes sociales; permitirá a sus usuarios gestionar videos, exponerlos en su muro, comentar videos propios y de sus amigos, calificarlos y compartirlos en varios canales.

En esta etapa del proyecto BananaTube se está abriendo a nuevas posibilidades, como que sus datos y servicios, sean consumidos por terceros y por apps modernas.

Para ello es imprescindible generar una buena interfaz que permita ofrecer los datos y servicios con consistencia, calidad y seguridad. Asimismo queremos que la interfaz esté abierta a extensiones de funcionalidad futuras, sin cambiar las que ya se estén consumiendo en ese momento.

Asimismo, queremos poder usar las potencialidades de las nuevas tecnologías para la implementación de APIs Restful usando la tecnología que ya conocemos: javascript.

Discutamos

- Qué características tiene una buena interfaz de consumo de datos/servicios?
- Qué principios deberíamos seguir para diseñarla?
- Qué formato nos permitirá ser interoperables y dinámicos?
- Cómo podemos garantizar la seguridad de la interfaz?
- Cómo podemos garantizar su correcta evolución?

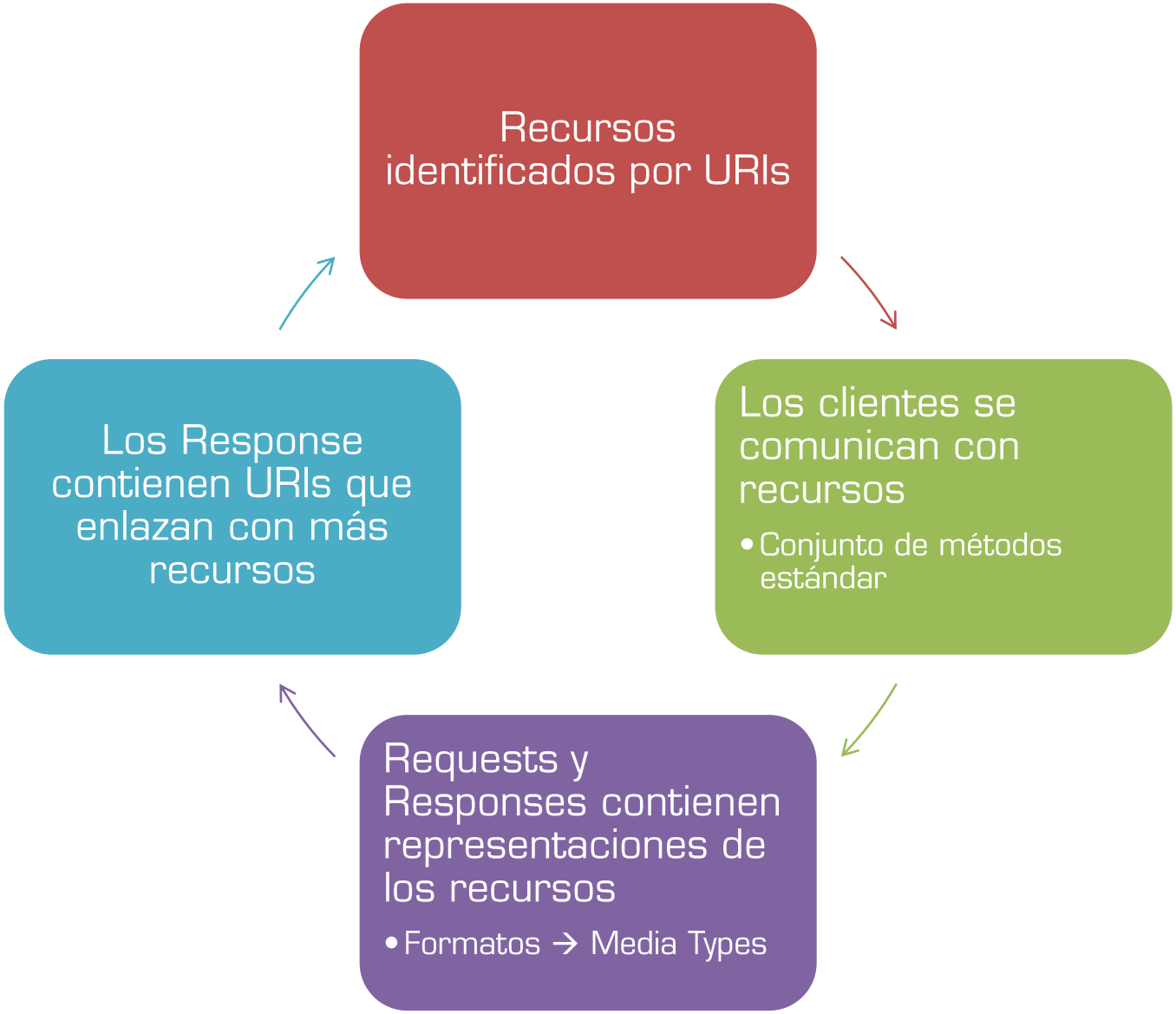
- Podemos usar javascript para implementar la API RESTful?
- Qué opciones tenemos?
- Qué arquitectura sería la más conveniente?

2 LOS PRINCIPIOS RESTFUL

REST

- REST: Representational State Transfer
- REST es una arquitectura para WWW
  - NO es una tecnología o estándar
  - Son un conjunto de Mejores prácticas
- Se basa en el protocolo HTTP (comandos, seguridad)
- Loosely-coupled, ligero y rápido de implementar
- Usa recursos eficientemente

Ciclo de las aplicaciones REST



Principios REST

- Dar a cada cosa un ID
- Conjunto de métodos estándar
- Vincular las cosas
- Múltiples representaciones
- Comunicaciones SIN estado

Dar a cada cosa un ID

- El ID de cada recurso es una URI

```
http://example.com/customers/1234
http://example.com/orders/2007/10/776654
http://example.com/products/4554
http://example.com/processes/salary-increase-234
```

- Los recursos son identificados por un ID
- Recursos == Clase Java
- Los ID se definen mediante anotación @Path

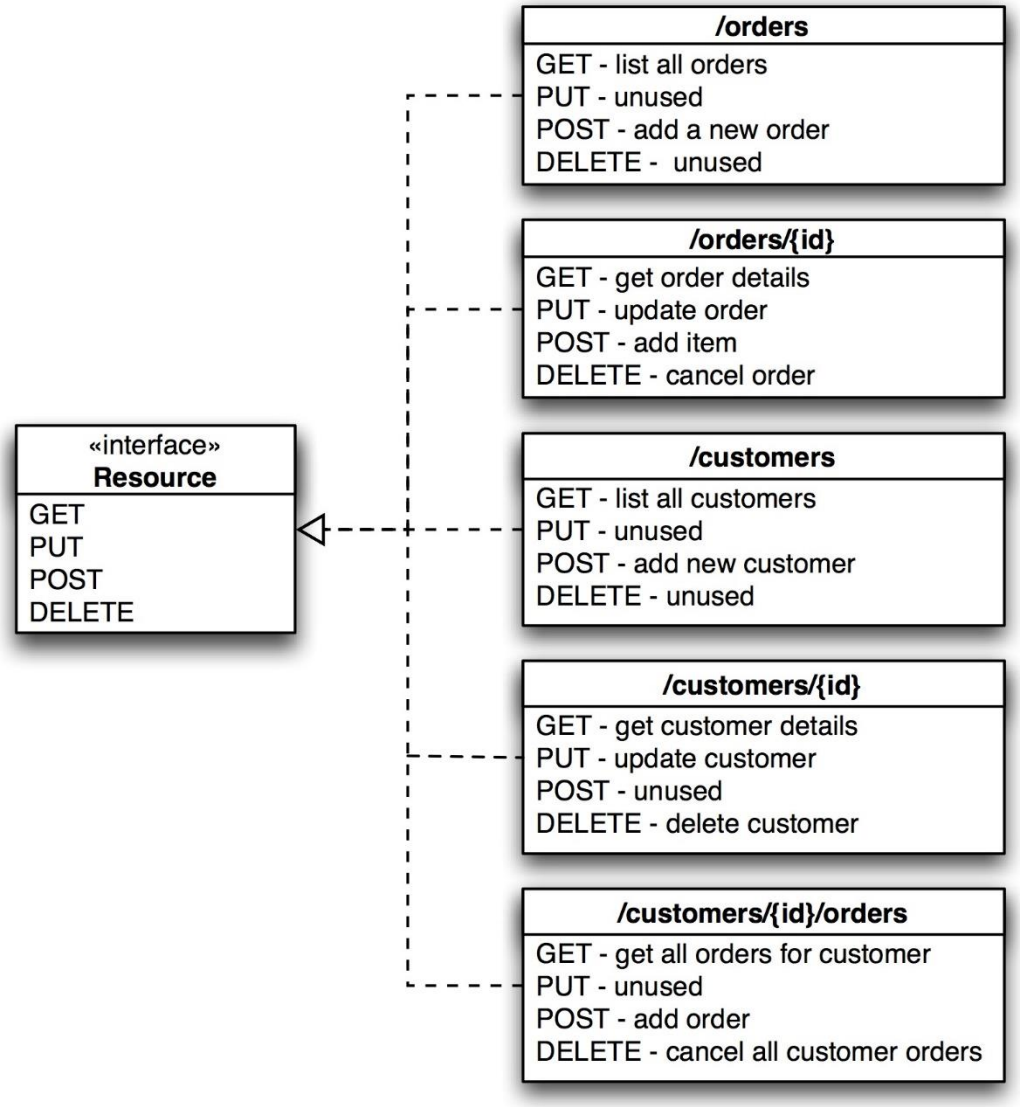
Conjunto de métodos estándar

- REST usa los comandos HTTP: PUT, DELETE, HEAD y OPTIONS

Método	Propósito	Anotación
GET	Leer (puede ser cacheado)	@GET
POST	Actualizar o crear (No idempotente)	@POST
PUT	Actualizar o crear (idempotente)	@PUT
DELETE	Borrar, eliminar	@DELETE
HEAD	GET sin Response	@HEAD
OPTIONS	Métodos soportados	@OPTIONS

- **Idempotencia**: la capacidad que tenga un ente (este caso el método) de realizar una misma operación varias veces, y obtener el mismo resultado que si solo se hiciese una vez.

Conjunto de métodos estándar (II)



Múltiples representaciones

- Ofrece datos en una variedad de formatos
  - XML, JSON, (X)HTML
- Maximizar el alcance
- Soportar el negociado de contenido
  - Aceptar cabecera

```
GET /foo
Accept: application/json
```

  - Basada en URI

```
GET /foo.json
```

Representación de recursos - MIMEs

- XML → application/properties+xml
- JSON → application/properties+json
- (X)HML+microformatos → application/html+xml

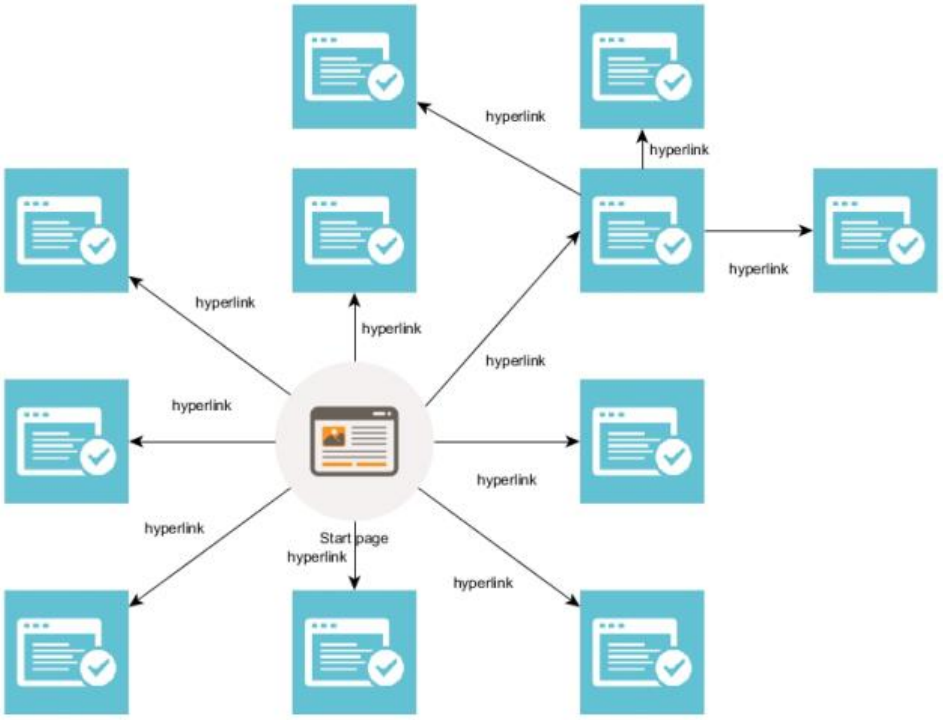
Vincular las cosas

- Los Response contienen links
- Una aplicación puede interpretar los vínculos (links) para obtener más información.
- La belleza del enfoque de los vínculos con las URIs es que los enlaces (links) pueden apuntar a los recursos que son ofrecidos por una aplicación diferente.
- Concepto HATEOAS: Hypermedia as the engine of application state

```
{
 "stocklist": {
  "name": "ACME",
  "price": "10.00",
  "link": [
   {
    "rel": "self",
    "href": "/stock/ACME",
    "method": "get"
   },
   {
    "rel": "buy",
    "href": "/account/ACME/buy",
    "method": "post"
   },
   {
    "rel": "sell",
    "href": "/account/ACME/sell",
    "method": "post"
   }
  ]
 }
}
```

HATEOAS (Hypermedia as the engine of application state)

"The name 'Representational State Transfer' is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."-Roy Fielding
Architectural Styles and the Design of Network-based Software Architectures - Chapter 6

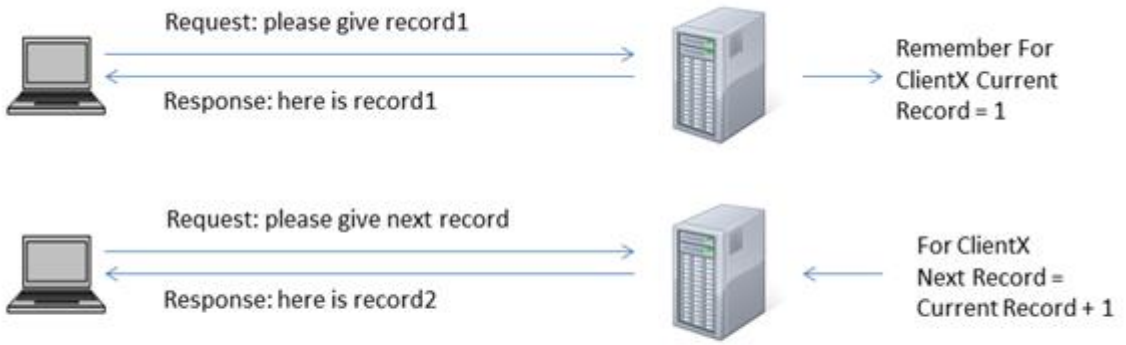


Comunicaciones SIN estado

- Los Identificadores van en la petición → independiente de la siguiente
- Todo lo que sea necesario procesar en el Request debe estar contenido en el Request
- REST exige que el estado sea transformado en estado del recurso y sea mantenido en el cliente.
- En otras palabras, un servidor no debería guardar el estado de la comunicación de cualquiera de los clientes que se comunican con él más allá de una petición única.
- La razón más obvia de esto es la escalabilidad - el número de clientes que pueden interactuar con el servidor se vería significativamente afectada si fuera necesario mantener el estado del cliente.

Diseño Sin Estado

- Sin REST



A Stateful Server – The server needs to remember the current record held by the client.

- Con REST

A Stateless Server– The server does not need to remember which record is being held by the client

3 DISEÑO Y ESPECIFICACIÓN DE APIS REST

API REST

- API es que es la abreviatura de *Application Programming Interface*, o *Interfaz de Programación de Aplicaciones*
- **Una API consigue que los desarrolladores interactúen con los datos de la aplicación de un modo planificado y ordenado**
- Una API REST es una interfaz de acceso a los servicios de una aplicación web.
- Diseñar una API es un proceso que hay que seguir con cuidado ya que será usada por aplicaciones clientes y por terceras partes.
- El objetivo del diseño es lograr una API que sea fácil de usar, fácil de adoptar y lo suficientemente flexible para implementarla en nuestras propias interfaces de usuario.

Requisitos fundamentales para una API

- Debe utilizar estándares web donde tengan sentido
- Debe ser amigable para el desarrollador y ser explorable mediante una barra de direcciones del navegador
- Debe ser sencilla, intuitiva y consistente para que la adopción no sólo sea fácil, sino también agradable
- Debe proporcionar suficiente flexibilidad para potenciar mayoritariamente la UI
- Debe ser eficiente, manteniendo el equilibrio con los demás requisitos
- Una API es una interfaz de usuario (UI) para un desarrollador – al igual que cualquier UI, es importante asegurar que la experiencia del usuario esté pensada cuidadosamente!

Uso correcto de URIs

- Las URL, Uniform Resource Locator , son un tipo de URI, Uniform Resource Identifier, que además de permitir **identificar de forma única el recurso**, nos permite localizarlo para poder acceder a él o compartir su ubicación.
- Una URL se estructura de la siguiente forma:

{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

Reglas básicas para URI de un recurso

Regla	Incorrecto	Correcto
Los nombres de URI no deben implicar una acción, por lo tanto debe evitarse usar verbos en ellos.	/facturas/234/editar	/facturas/234 para editar, borrar, consultar
Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.		
Deben ser independiente de formato.	/facturas/234.pdf	/facturas/234 para pdf, epub, txt, xml o json
Deben mantener una jerarquía lógica.	/facturas/234/cliente/007	/clientes/007/facturas/234
Los filtrados de información de un recurso no se hacen en la URI.	/facturas/orden/desc/fecha-desde/2007/pagina/2	/facturas?fecha-desde=2007&orden=DESC&tpagina=2

Usa acciones y URLs RESTful

- Estos recursos son manipulados usando peticiones HTTP donde el método (GET, POST, PUT, PATCH, DELETE) tienen un significado específico.
- Ejemplos:
  - **GET /facturas** Nos permite acceder al listado de facturas
  - **POST /facturas** Nos permite crear una factura nueva
  - **GET /facturas/123** Nos permite acceder al detalle de una factura
  - **PUT /facturas/123** Nos permite editar la factura, sustituyendo la totalidad de la información anterior por la nueva.
  - **DELETE /facturas/123** Nos permite eliminar la factura
  - **PATCH /facturas/123** Nos permite modificar cierta información de la factura, como el número o la fecha de la misma.

Códigos de estado

- Un error común es no devolver el código de estado que indique la respuesta (https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP)

- **Incorrecto**: el código es 200, pero en el cuerpo de respuesta indicamos un error

```
Petición
========
PUT /facturas/123

Respuesta
=========
Status Code 200
Content:
{
  success: false,
  code:    734,
  error:   "datos insuficientes"
}
```

- **Correcto**: el código es 400, que se corresponde con el error del cuerpo

```
Petición
========
PUT /facturas/123

Respuesta
=========
Status Code 400
Content:
{
  message: "se debe especificar un id de cliente para la factura"
}
```

Más recomendaciones

- Usa SSL en todos lados, sin excepciones
- Una API es tan buena como lo es su documentación – por lo tanto realiza una buena documentación
- HATEOAS todavía no es muy práctico
- Usa JSON donde sea posible, XML sólo si tienes la obligación
- Deberías usar camelCase con JSON, pero snake_case es un 20% más fácil de leer
- Considera usar JSON para cuerpos de peticiones POST, PUT y PATCH
- Usa autenticación basada en tokens, transportado en OAuth2 donde se necesite delegación
- Utilizar efectivamente los códigos de error HTTP

La API de BananaTube

- Siguiendo el ejemplo del sitio web
  - https://www.paradigmadigital.com/dev/definir-prototipar-api-rest/

- Define la API de BananaTube usando la herramienta **Swagger**
  - http://editor.swagger.io/#/

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"