<XML> VS {JSON} VS YAML

<XML> VS {JSON} VS YAML

Extended Markup Lenguage

Javascript Object Notation

Yet another markup lenguage → YAML Ain’t Markup Language

```
<clientes>
    <cliente id="1">
        <nombre>Ringo</nombre>
        <direccion>bbb</direccion>
        <telefono>ccc</telefono>
    </cliente>
    <cliente id="1">
        <nombre>Paul</nombre>
        <direccion>bbb</direccion>
        <telefono>ccc</telefono>
    </cliente>
    <cliente id="1">
        <nombre>George</nombre>
        <direccion>bbb</direccion>
        <telefono>ccc</telefono>
    </cliente>
    <cliente id="1">
        <nombre>John</nombre>
        <direccion>bbb</direccion>
        <telefono>ccc</telefono>
    </cliente>
</clientes>
```

```
[
    {
        "nombre" : "Ringo",
        "direccion" : "C/Falsa, 123",
        "telefono" : "555"
    },
    {
        "nombre" : "Paul",
        "direccion" : "C/Falsa, 123",
        "telefono" : "555"
    },
    {
        "nombre" : "George",
        "direccion" : "C/Falsa, 123",
        "telefono" : "555"
    },
    {
        "nombre" : "John",
        "direccion" : "C/Falsa, 123",
        "telefono" : "555"
    },
]
```

```
clientes :
  - cliente :
      nombre : "Ringo"
      direccion : "C/Falsa, 123"
      telefono : "555"
  - cliente :
      nombre : "Paul"
      direccion : "C/Falsa, 123"
      telefono : "555"
  - cliente :
      nombre : "George"
      direccion : "C/Falsa, 123"
      telefono : "555"
  - cliente :
      nombre : "John"
      direccion : "C/Falsa, 123"
      telefono : "555"
```

PROTOCOLO HTTP

En un servidor remoto (al que envía la petición) hay una serie de recursos organizados en carpetas (reales o imaginarias) Las aplicaciones cliente envían peticiones para trabajar con dichos recursos

Características del protocolo:

- De texto
- Sin estado

Estructura de una petición

Una petición http es la suma de una ruta a uno o varios recursos y un método que indica qué hay que hacer con ellos

Métodos HTTP

Métodos que afectan directamente a los recursos

- GET: Solicita recursos existentes en el servidor. No tiene body
- POST: Solicita al servidor que añada un recurso. Body opcional
- PUT: Solicita la sustitución de un recurso por otro. Body opcional
- PATCH: Solicita la modificación de un recurso. Body opcional
- DELETE: Solicita al servidor eliminar uno o varios recursos. No tiene body

Métodos para otras cosas

- HEAD
- OPTIONS
- ...

Rutas

Las rutas incluyen el caracter '/' para representar las carpetas en las que están alojados los recursos

Peticiones

METODO + RUTA

```
get /inicio.html
get /paginas/catalogo.html
get /estilos/estilos.css
```

REST

Representational State Transfer (ReST)

Ideado por Thomas Roy Fielding

Pasos a seguir para diseñar un API REST

Definir los recursos

Aplicación de gestión de una empresa

- Usuarios
- Clientes

Comerciales

- Empleados
- Facturas
- DetallesFactura
- Productos
- Empleados
- …

Otro ejemplo, el de IMDB

- Películas
- Valoraciones
- Premios
- Localizaciones
- Actores
- Directores
- Series
- Episodios
- Generos

Cada recurso tendrá una serie de valores

Película:

- id
- título
- duracion
- fecha de estreno
- Calificación
- …

Los recursos podrán estar relacionados entre si y esas relaciones tendrán cardinalidad

- director (1..1)
- actores (n..m)
- Generos (1..n) siendo el primero de la lista el género principal ó
- generoPrincipal 1..1
- generosSecundarios 1..n

Decidir el identificador de los recursos

Los recursos se identificarán por un valor que será único

- Id
- Código
- Referencia
- ...

Definir en qué carpetas estarán alojados los recursos

- Como norma las carpetas estarán en plural.
- Se contemplan casos excepcionales en los que se espera encontrar un único recurso en la carpeta y por lo tanto es legal ponerla en singular
- por convenio se escriben en minusculas
- si el nombre del recurso es más de una palabra se utiliza camelCase por defecto aunque no es ninguna ley
- categoriasProducto (camel case)
- categorias-producto (kebak case)
- categorias_producto (snake case)

```
/clientes
    cli-1
    cli-2
    cli-4

/facturas
    fac-001
    fac-002
    fac-003
```

Asociar peticiones a las funcionalidades que ofrece el servicio

Tendremos rutas que identifican unicamente a un recurso y rutas que identificarán a varios

- La ruta que acaba en carpeta identifica a todos los recursos contenidos en ella
- La ruta que acaba en un identificador se refiere a solo un recurso

GET	/clientes	Listar clientes
GET	/clientes/	Buscar cliente por id
POST	/clientes	Insertar cliente
DELETE	/clientes	Eliminar todos los clientes !!!
DELETE	/clientes/cli-3	Eliminar un cliente

Definir las respuestas

Decidir el formato en el que estará la información

- XML
- JSON
- YAML
- CSV
- Uno inventado
- ...

Lo normal es escoger entre XML y JSON o dejar que la aplicación cliente escoja entre esos dos

Decidir de qué manera se representarán los recursos (mas allá del formato)

> **NOTE** | El mismo recurso podrá representarse de distintas maneras en distintas situaciones

Adjuntar a la respuesta el STATUS CODE correcto según el protocolo HHTP

- 1XX : Respuestas informativas que no ha usado nadie en la vida
- 2XX : Respuestas satisfactorias. Todo fue bien
- 3XX : Redirecciones. Vamonos
- 4XX : Error en la petición. Te has equivocado tú
- 5XX : Error en el servidor. No shemos equivocado nosotros

Query parameters

- Con los query filtramos los resultados de una petición GET
- Afinamos los recursos identificados por la ruta
- Los parámetros se añaden a continuacion de la ruta, despues de una interrogación
- NO forman parte de la ruta
- Es como el WHERE de un SELECT

Ejemplo: Queremos un listado de clientes. Enviamos un GET

```
GET /clientes
```

Esto significa 'dame todos los clientes' y si son muchos no tiene sentido. Podemos enviar criterios de filtrado para indicar cuales recursos se solicitan

```
GET /clientes?ciudad=Chinchón
GET /clientes?estado=activo
GET /clientes?ciudad=Chinchón&estado=activo
```

Un parámetro por defecto significa 'igual a'

```
GET /peliculas?tituloIgualA="Alien"
GET /peliculas?titulo="Alien" <--- Mucho mejor esta
```

Los parámetros se pueden utilizar con otros métodos HTTP Ejemplo: borrar todos los clientes de Chinchón:

```
DELETE /clientes?ciudad=Chinchón
```

Parametros complejos

Obtener productos entre dos precios

```
GET /productos ?precioIgualA=100
GET /productos ?precioMin=100
GET /productos ?precioMax=200
GET /productos ?precioMin=100&precioMax=200
```

Podemos uniformizar este tipo de parámetros (esto no forma parte del protocolo http)

```
GET /productos?precio=100
GET /productos?precio=eq:100
GET /productos?precio=lt:100
GET /productos?precio=gt:100
GET /productos?precio=ne:100
```

Definiendo un API REST

Supongamos que los recursos son clientes

Funcionalidades:

- Ver detalle cliente
- Listar clientes
- Alta cliente
- Modificar cliente

- Baja cliente

Ver detalle cliente

Petición:

```
GET /clientes/{identificador}
```

El identificador es una parte variable en la ruta

Posibles respuestas:

```
200 OK
Content-Type: application/json
--------------------------------
{
    "id"        : 1,
    "nombre"    : "Ringo Starr",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456"
}
```

```
404 NOT FOUND
Content-Type: application/json
--------------------------------
{
    "codigo"  : 404,
    "mensaje" : "El cliente no existe"
}
```

```
500 INTERNAL SERVER ERROR
Content-Type: application/json
--------------------------------
{
    "codigo"  : 500,
    "mensaje" : "Hubo un problema en el servidor"
}
```

Listar clientes

Petición:

```
GET /clientes/{identificador}
```

Posibles respuestas:

```
200 OK
Content-Type: application/json
--------------------------------
[{
    "id"        : 1,
    "nombre"    : "Ringo Starr",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456"
}]
```

Si no hay clientes tenemos dos opciones:

1: 404

```
404 NOT FOUND
Content-Type: application/json
--------------------------------
{
    "codigo"  : 404,
    "mensaje" : "No hay clientes"
}
```

2: 200

```
200 OK
Content-Type: application/json
--------------------------------
[]
```

```
500 INTERNAL SERVER ERROR
Content-Type: application/json
--------------------------------
{
    "codigo"  : 500,
    "mensaje" : "Hubo un problema en el servidor"
}
```

Alta cliente

Petición:

- POST
- Un post incluye una ruta que termina en carpeta
- El recurso a añadir va en el body

```
POST /clientes
Content-Type: application/json
------------------------------
{
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456"
}
```

Posibles respuestas:

```
201 CREATED
Content-Type: application/json
------------------------------
{
    "codigo"  : 201,
    "mensaje" : "El cliente se ha dado de alta"
}
```

También podemos devolver el recurso tal cual ha quedado en el servidor:

```
201 CREATED
Content-Type: application/json
------------------------------
{
    "id"        : 123,
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456",
    "sucursal"  : "San Francisco"
}
```

Error de validación:

```
400 BAD REQUEST
Content-Type: application/json
------------------------------
{
    "codigo"  : 400,
    "mensaje" : "Error de validación. Nombre es obligatorio"
}
```

500, como en las anteriores.

Modificar cliente

Podemos hacerlo con PUT y con PATCH

Con PUT:

```
PUT /clientes/123
Content-Type: application/json
-------------------------------
{
    "id"        : 123, <--Ver nota
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle, nº 42",
    "telefono"  : "555123456",
}
```

NOTE	El id es obligatorio en la ruta y opcional en el json que adjuntamos al body. Si viene en los dos lugares el servidor debe de asegurarse de estar utilizando el id de la ruta

PUT es sustituir un recurso por otro. La siguiente petición, si no s ponemos tiquismiquis, significa que queremos dejar al cliente 123 solo con la direccion (y el id).

Depende de como esté programado el servidor el que suceda o no. Lo suyo es que se valide y se devuelva un 400

```
PUT /clientes/123
Content-Type: application/json
-------------------------------
{
    "direccion" : "C/Su calle, nº 42",
}
```

Con PATCH:

A diferencia de PUT (sustituir recurso) PATCH es modificar el recurso

Supongamos que tenemos el siguiente cliente en la BB.DD.:

```
{
    "id"        : 123,
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456",
    "sucursal"  : "San Francisco"
}
```

Si procesamos la siguiente petición:

```
PATCH /clientes/123
Content-Type: application/json
-------------------------------
{
    "direccion" : "C/Su calle, nº 42",
}
```

entonces el recurso quedaría así:

```
{
    "id"        : 123,
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle, nº 42",
    "telefono"  : "555123456",
    "sucursal"  : "San Francisco"
}
```

Si hubiera sido con PUT:

```
{
    "id"        : 123,
    "nombre"    : null,
    "direccion" : "C/Su calle, nº 42",
    "telefono"  : null,
    "sucursal"  : null
}
```

Posibles respuestas:

```
200 OK
Content-Type: application/json
------------------------------
{
    "codigo"  : 201,
    "mensaje" : "El cliente se ha modificadoa"
}
```

También podemos devolver el recurso tal cual ha quedado en el servidor:

```
200 OK
Content-Type: application/json
------------------------------
{
    "id"        : 123,
    "nombre"    : "Harry Callahan",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456",
    "sucursal"  : "San Francisco"
}
```

Error de validación:

```
400 BAD REQUEST
Content-Type: application/json
------------------------------
{
    "codigo"  : 400,
    "mensaje" : "Error de validación. Nombre es obligatorio"
}
```

500, como en las anteriores.

Baja cliente

Petición:

```
DELETE /clientes/:id
```

Posibles respuestas:

```
200 OK
Content-Type: application/json
--------------------------------
{
    "codigo" : 200,
    "mensaje" : "El cliente se eliminó"
}
```

Si no existe un cliente con ese identificador:

1: 404

```
404 NOT FOUND
Content-Type: application/json
--------------------------------
{
    "codigo"  : 404,
    "mensaje" : "El cliente no existe"
}
```

500, como en las anteriores

Modificaciones y borrados a granel

Bulk updates/Bulk deletes

Supongamos que tenemos los recursos 'producto' en la carpeta '/productos'

Modificar un producto concreto (con patch sería igual, y condelete DELETE /productos/45):

```
PUT /productos/45
Content-Type: application/json
-----------------------------------------
{
    "nombre" : "Fleje",
    "fabricante" : "Ibérica de flejes S.A."
    ...
}
```

Si con los métodos PUT, PATCH o DELETE no indicamos el identificador al final de la ruta estaremos identificando varios recursos a la vez:

Cambiar las existencias de tododos los productos a 1.000:

```
PATCH /productos
Content-Type: application/json
-----------------------------------------
{
    "existencias" : 1000
}
```

Borrar todos los productos:

```
DELETE /productos
```

Borrar todos los productos que no tengan existencias y que no se hayan comprado en un año:

Si tenemos esta tabla en la base de datos:

```
PRODUCTOS

ID  | NOMBRE | FABRICANTE | EXISTENCIAS | FECHA_ULTIMA_COMPRA
-------------------------------------------------------------
1   | Fleje  | IDF S.A.   | 0           | 1/1/2021
```

Petición:

```
DELETE /productos?existencias=0&ultimaCompra=lt:23/03/2021
```

Si tuvieramos este otro diseño en la base de datos:

```
PRODUCTOS

ID  | NOMBRE | FABRICANTE
-------------------------
25   | Fleje  | IDF S.A.

EXISTENCIAS_PRODUCTO

ID  | EXISTENCIAS
-----------------
25   | 0

PEDIDOS

ID | CODIGO | CLIENTE | FECHA
---------------------------------
42 | PED-42 | 15      | 01/01/2021

DETALLES_PEDIDOS

ID_PEDIDO | ID_PRODUCTO | CANTIDAD
---------------------------------
42        | 25          | 5
```

Entonces la petición es IGUAL:

```
DELETE /productos?existencias=0&ultimaCompra=lt:23/03/2021
```

Relaciones entre recursos

Tenemos relaciones entre las tablas de la bb.dd., entre las clases del proyecto y también entre los recursos de los apis Rest.

Podemos expresarlas en las rutas

Tenemos clientes y facturas. Un cliente tiene muchas facturas y una factura pertenece a un cliente. Es una relación de cardinalidad 1..n

```
/clientes
    cli-1
    cli-2
    cli-4

/facturas
    fac-001
    fac-002
    fac-003
    fac-004
    fac-005
    fac-006
```

Si nos pidieran las facturas de un cliente podríamos haber organizado los directorios así:

```
/facturas
    /clientes
        /cli-1
            fac-1
            fac-4
        /cli-2
            fac-3
            fac-5
        /cli-3
            fac-3
            fac-6
```

Esto está mal por que la relación es 'los clientes tienen facturas' no 'las facturas tienen cliente' Además, según el protocolo HTTP los recursos que se identifican por la ruta son los que están en la última carpeta:

```
GET /facturas/clientes/cli-1 <--Esto identifica al cliente 1, no a sus facturas
```

Mucho mejor esta otra organizacion

```
/clientes
    /cli-1
        cli-1
        /facturas
            fac-1
            fac-4
        /albaranes
        /incidencias
    /cli-2
        cli-1
        /facturas
            fac-3
            fac-5
        /albaranes
        /incidencias
    /cli-3
        cli-1
        /facturas
            fac-3
            fac-6
        /albaranes
        /incidencias
```

Ahora disponemos de las siguientes posibilidades (si lo implementamos en el servidor):

```
GET /clientes                -> todos los clientes
GET /clientes/cli-1          -> el cliente 1
GET /clientes/cli-1/facturas -> las facturas del cliente 1
```

En cambio esta peticion...

```
GET /clientes/cli-1/facturas/fac-5
```

Se debería haber definido así (a no se que las facturas tuvieran una numeración que depende del cliente):

```
GET /facturas/fac-5
```

Dame las peliculas de un director

```
GET /directores/dir-5/peliculas
```

Relaciones bidireccionales

Las relaciones bidireccionales de n..m tambien se ven reflejadas en las rutas a los recursos. Eneste ejemplo las dos opciones tienen significado aunque sea uno distinto

Dame los actores de una pelicula

```
GET /peliculas/peli-1/actores
```

Dame las películas de un actor

```
GET /actores/actor-1/peliculas
```

No todo en esta vida se puede hacer con carpetas y relaciones entre recursos

Dame las peliculas de 1984:

```
GET /años/1984/peliculas
```

Año (es este api imaginario) no es un recurso asi que no puede ser una carpeta. Aqui año es una propiedad de los recursos 'película'

```
GET /peliculas?año=1984
```

Nadie nos obliga a indicar las relaciones entre los recursos con carpetas en la ruta

Esta peticion

```
GET /clientes/{idCliente}/facturas
```

Significa exactamente lo mismo que esta:

```
GET /facturas?idCliente={idCliente}
```

Y las dos son correctas.

No todo en esta vida es un CRUD

Tenemos un recurso 'pedido' en la carpeta '/pedidos'

El recurso tiene estas propiedades

```
{
    "id"      : Entero,
    "codigo"  : String,
    "cliente" : X
    "fecha"   : String,
    "estado"  : CREADO|ACEPTADO|CANCELADO
}
```

Tenemos este api para el CRUD:

```
GET    /pedidos?    : listar pedidos por criterio
GET    /pedidos/:id : buscar un pedido por su identificador
POST   /pedidos     : insertar un pedido
PUT    /pedidos/:id : modificar un pedido
DELETE /pedidos/:id : borrar un pedido
```

Aceptar un pedido:

```
PUT /pedidos/:id
Content-Type: application/json
------------------------------
{
    "estado" : "ACEPTADO"
}
```

Una petición como esta no sirve porque ya hemos asociado 'PUT /pedidos/:id' a modificar pedido.

IMPORTANT	No debe existir ninguna ambiguedad a la hora de procesar peticiones. No podemos tener que 'PUT /pedidos/:id' sirva para modificar un pedido y para aceptarlo.

En el código del servidor se ve claramente:

```
@RestController
public class ClientesRest {

    @Autowired private GestorPedidos gestorPedidos;

    @PutMapping(path="/pedidos/{id}")
    public void modificarAceptar(@PathVariable("id") Integer id, @RequestBody() Pedido pedido)
{

        GestorPedidos gp = new GestorPedidos();

        //OH DIOS MIO, UN IF Y UN ELSE!
        //NO ESTAMOS RESPETANDO EL PRINCIPIO DE RESPONSABILIDAD ÚNICA
        //No respetar ese principio es no respetarse a si mismo

        //Se ve claramente como este método hace dos cosas

        if(pedido.getEstado().equals("ACEPTADO")) {
            gp.aceptarPedido(id);
        } else {
            gp.modificarPedido(pedido);
        }
    }
}

class GestorPedidos {

    public void aceptarPedido(Integer id) {
        //Lógica de negocio para aceptar
    }

    public void modificarPedido(Pedido pedido) {
        //Lógica de negocio para modificar
    }

}

class Pedido {
    private Integer id;
    private String estado;

    public Pedido() {
        super();
    }

    ...
}
```

Segundo intento

Lo que queremos en realidad es ejecutar una acción sobre un recurso. Estos ejemplos no necesitan body porque en la ruta ya está el identificador

```
PUT /pedidos/:id?operacion=ACEPTAR
PUT /pedidos/:id?accion=ACEPTAR
PUT /pedidos/:id?comando=ACEPTAR
```

Seguimos mal. Esto no es REST

- Primero: en la implementación del servidor seguimos teniendo un método que sirve para dos cosas:

```
    @PutMapping(path="/pedidos/{id}")
    public void modificarAceptar(
            @PathVariable("id") Integer id,
            @RequestBody() Pedido pedido,
            @RequestParam(name="operacion", required=false) String operacion
        ) {

        GestorPedidos gp = new GestorPedidos();
        if("ACEPTAR".equals(operacion)) {
            gp.aceptarPedido(id);
        } else {
            gp.modificarPedido(pedido);
        }
    }
```

- En el protocolo HTTP los únicos verbos admitidos son GET, POST, PUT, PATCH y DELETE y nosotros estamos añadiendo uno de manera artificial en forma de parámetro

Lo mismo se aplicaría a:

```
PUT /pedidos/:id/acciones/aceptar
PUT /pedidos/:id/aceptar
PUT /aceptarPedido/:id
```

No vale hacer trampa, 'ocultar' el verbo es igual de malo

```
PUT /pedidos/:id/aceptacion
```

Tampoco cuela ocultar el verbo en un header

```
PUT /pedidos/:id
Operacion: ACEPTAR
```

Solución ideal

La solución ideal sería que el protocolo permitiera la creación de métodos personalizados, pero tal cosa no existe:

```
GET     /pedidos?     : listar pedidos por criterio
GET     /pedidos/{id} : buscar un pedido por su identificador
POST    /pedidos      : insertar un pedido
PUT     /pedidos/{id} : modificar un pedido
DELETE  /pedidos/{id} : borrar un pedido

ACEPTAR /pedidos/{id} : aceptar un pedido
```

Lo que debemos hacer es inventarnos un nuevo recurso porque no podemos inventarnos un nuevo método

Si ninguno de los métodos HTTP al aplicarlos a nuestro recurso significan lo que necesitamos, entonces nos inventamos un recurso tal que al aplicarle GET/POST/PUT/PATCH/DELETE tenga el significado correcto

Por ejemplo: no queremos aceptar un pedido: queremos crear una orden de compra

```
POST /ordenesCompra
ContentType: application/json
-----------------------------
{
    "idPedido" : 42
}
```

NOTE	:)

Hiperenlaces

Petición:

```
GET /clientes/78
```

Respuesta:

Aqui estamos adjuntando a los datos del cliente la dirección y las facturas. Tan malo será dar de menos (nos harán más peticiones) que dar de más (hemos hecho un gasto de recursos inutil)

```
200 OK
Content-Type: application-json
------------------------------
{
    "id"        : 78
    "nombre"    : "Bud Spencer",
    "direccion" : {
            "ciudad" : "Gran Alacant, Santa Pola",
            "calle"  : "Creta"
            "numero" : "123"
        },
    "telefono"  : "555"
    "facturas"  : [
            {
                "codigo" : "FAC-10"
                ...
            },
            {
                "codigo" : "FAC-88"
                ...
            }
        ]
}
```

Esto es más razonable. Si quieren las facturas que nos envíen un 'GET /facturas?idCliente=78'

```
200 OK
Content-Type: application-json
------------------------------
{
    "id"        : 78
    "nombre"    : "Bud Spencer",
    "direccion" : {
            "ciudad" : "Gran Alacant, Santa Pola",
            "calle"  : "Creta"
            "numero" : "123"
        },
    "telefono"  : "555"
}
```

Hiperenlaces

Se incluyen en el recuso enlaces a los recursos relacionados

- Se da por sentado que al seguir el enlace será una petición GET
- Se incluye un enlace al própio recurso

```
200 OK
Content-Type: application-json
------------------------------
{
    "id"       : 78
    "nombre"   : "Bud Spencer",
    "telefono" : "555"

    "enlaces"  : [
            {
                "self" : "/clientes/78"
            },
            {
                direccion : "/direcciones?idCliente=78" | Válida
                direccion : "/clientes/78/direcciones"  | Mejor
                direccion : "/clientes/78/direccion"    | Legal
            },
            {
                facturas : "/clientes/78/facturas"  |
                facturas : "/facturas?idCliente=78" | Las dos son válidas
            },
            {
                incidencias : "/incidencias?idCliente=78"
            },
            {
                mensajes : "/mensajes?idCliente=78"          |
                mensajes : "/mensajesClientes?idCliente=78"  |
                mensajes : "/mensajes/clientes?idCliente=78" | La que sea
            },
        ]
}
```

Si hemos puesto que la dirección sea un enlace forzaremos a las aplicaciones cliente a hacer una segunda petición simpre, porque la dirección siempre será necesaria. Pues adjuntamos la dirección en lugar del enlace:

```
200 OK
Content-Type: application-json
------------------------------
{
    "id"       : 78
    "nombre"   : "Bud Spencer",
    "telefono" : "555",
    "direccion" : {
            "ciudad" : "Gran Alacant, Santa Pola",
            "calle"  : "Creta"
            "numero" : "123"
        },
    "enlaces"  : [
            {
                "self" : "/clientes/78"
            },
            {
                "facturas" : "/facturas?idCliente=78"
            },
            {
                "incidencias" : "/incidencias?idCliente=78"
            },
            {
                "mensajes" : "/mensajes/clientes?idCliente=78"
            },
        ]
}
```

Estandarización del api

Tenemos esta petición:

```
GET /clientes/{id}
```

Respuestas:

```
200 OK
Content-Type: application/json
------------------------------
{
    "id"        : 1,
    "nombre"    : "Ringo Starr",
    "direccion" : "C/Su calle",
    "telefono"  : "555123456"
}
```

```
404 NOT FOUND
Content-Type: application/json
--------------------------------
{
   "codigo"  : 404,
   "mensaje" : "El cliente no existe"
}
```

A veces resulta conveniente que las respuestas sean uniformes, que el servidor responda siempre con lo mismo

Definimos un documento genérico para todas las respuestas:

```
{
    "status" : ""
    "error"  : {
            "codigo"   : "El que sea",
            "mensaje"  : "Nose qué",
            "detalles" : X
            ...
        }
    "datos" : {
            "descripcion" : "",
            "valor"       : X
            ...
        }
    "paginacion" : {
            "total-paginas"   :
            "recursos-pagina" :
            "numero-pagina"   :
        }
}
```

Si la respuesta es un 200:

```
200 OK
Content-Type: application/json
--------------------------------
{
    "status" : 200
    "datos"  : {
            "descripcion" : "cliente",
            "valor" : {
                "id"        : 1,
                "nombre"    : "Ringo Starr",
                "direccion" : "C/Su calle",
                "telefono"  : "555123456"
            }
        }
}
```

Si la respuesta es un error (4XX/5XX)

```
200 OK
Content-Type: application/json
--------------------------------
{
    "status" : 400
    "error"  : {
            "codigo" : "El que sea",
            "mensaja" : "Nose qué",
            ...
        }
}
```

Normnalización

Tenemos esta petición:

```
GET /clientes/1
```

Respuestas:

```
200 OK
Content-Type: application/json
--------------------------------
{
    "id"        : 1,
    "nombre"    : "Alien",
    "telefono" : "555123456"
    "direccion"  : {
            "calle" : "C/Tal"
        }
    "facturas"  : [
            { ... },
            { ... },
            { ... },
            { ... },
        ],
    "incidencias"  : [
            { ... },
            { ... },
            { ... },
            { ... },
        ]
}
```

Luego, en las aplicaciones cliente, se accedería a los valores de la respuesta de este modo:

```
cliente.nombre
cliente.getNombre()

cliente.direccion.calle
cliente.getDireccion().getCalle()

listaFacturas = cliente.facturas
listaFacturas = cliente.getFacturas()
```

Otra manera de hacerlo sería esta, en la que la información está normalizada

```
200 OK
Content-Type: application/json
--------------------------------
{
    "cliente" : {
        "id"       : 1,
        "nombre"   : "Alien",
        "telefono" : "555123456"
    },
    "direccion"  : {
            "calle" : "C/Tal"
        }
    "facturas"  : [
            { ... },
            { ... },
            { ... },
            { ... },
        ],
    "incidencias"  : [
            { ... },
            { ... },
            { ... },
            { ... },
        ]
}
```

Luego, en las aplicaciones cliente, se accedería a los valores de la respuesta de este modo:

```
respuesta.cliente.nombre
respuesta.getCliente.getNombre()

respuesta.cliente.direccion.calle
respuesta.getCliente().getDireccion().getCalle()

listaFacturas = respuesta.cliente.facturas
listaFacturas = respuesta.getCliente().getFacturas()
```