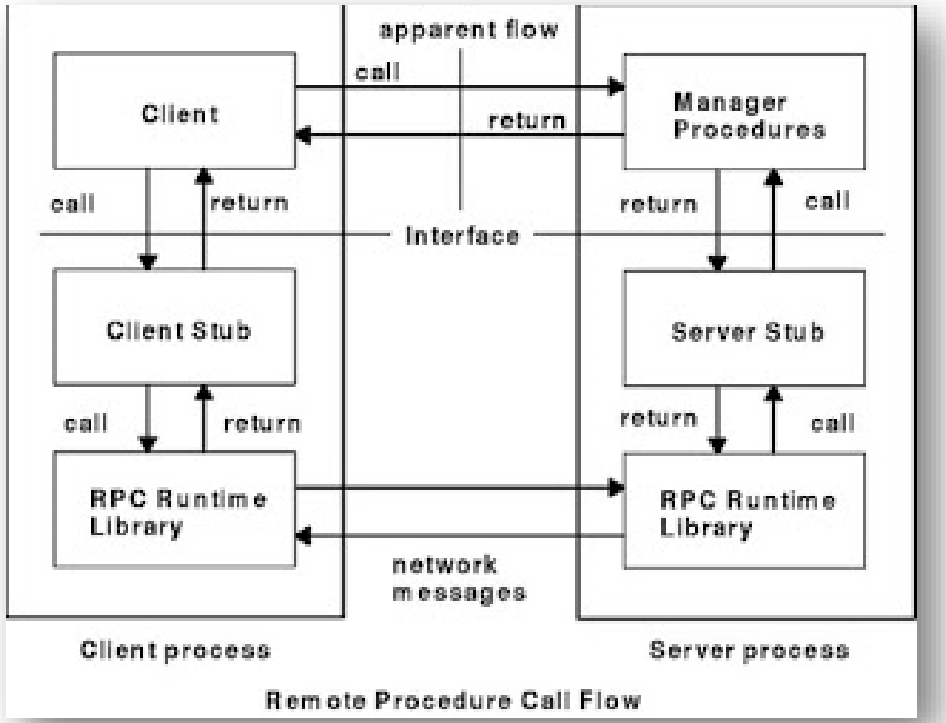
REST
Un estilo arquitectónico

ING. CARLOS FLORES ORIHUELA

RPC - REST



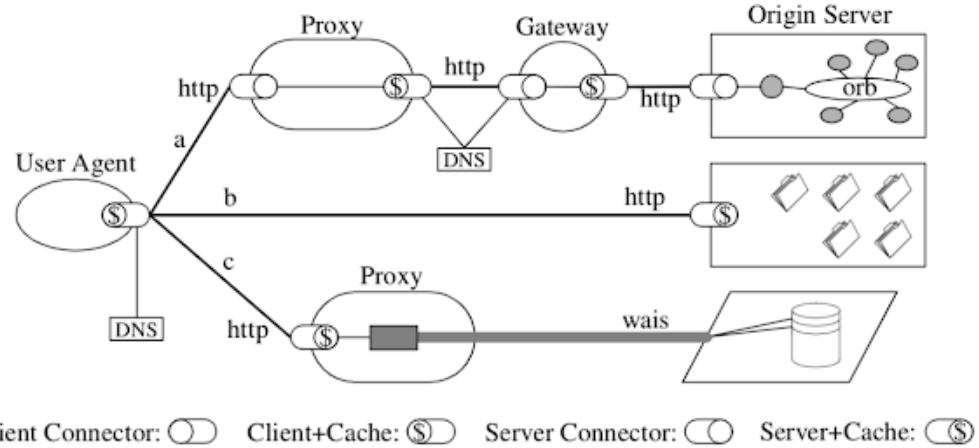


Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

REPRESENTATIONAL STATE TRANSFER

A finales de 1993, Roy Fielding, cofundador del Proyecto Apache HTTP Server, se preocupó por el problema de escalabilidad de la Web, Fielding reconoció que la escalabilidad de la Web se regía por un conjunto de ”*restricciones*” clave.

Fielding trabajó junto a Tim Berners-Lee y otros también para *una nueva* especificación del Protocolo de transferencia de hipertexto, HTTP / 1.1 y sintaxis de las Uniformes de Recursos (URI) en RFC 3986.

El año 2000 **Roy Fielding presenta su tesis doctoral titulada Architectural Styles and the Design of Network-based Software Architectures, University of California Irvine.**

REST se rige por un conjunto coordinado de restricciones que controlan el funcionamiento y las características de los elementos de la arquitectura y permiten las relaciones de unos elementos con otros.

REST

REST es un estilo arquitectónico centrado en recursos que hace uso de URI y verbos HTTP (por ejemplo, GET, PUT, POST, DELETE y PATCH) para realizar acciones contra un recurso (único) dado

RESTRICCIONES REST

1. CLIENT/SERVER: La Web es un sistema basado en cliente-servidor, server (software api middleware) / client, ambos deben **evolucionar independientemente**, el cliente sólo debe conocer la URI de recursos y eso es todo, mantenerlo simple alineado al Interfaz Uniforme Web, uri e hiperenlaces.
2. STATELESS/ Sin Estado (Apatridia):

   "*No se almacenará ningún contexto de cliente en el servidor entre solicitudes. El cliente es responsable de administrar el estado de la aplicación.*"

Ventajas: Escalabilidad (el servicio se libera rápidamente), mejora visibilidad (no ve otros), eficiencia (recuperación de errores), mejora la eficiencia x liberación de recursos.

Desventaja: Tráfico de Red.

Como el servidor no necesita administrar ninguna sesión, es posible implementar los servicios en cualquier número de servidores, por lo que la escalabilidad nunca será un problema.

El estado lo mantiene el cliente y por lo tanto es el cliente quien debe pasar el estado en cada llamada, eso puede ser un usuario y una contraseña, un *token* o cualquier otro tipo.

RESTRICCIONES REST

3. CACHEABLE: Cliente y Servidor: Idempotencia, Varnish, API Gateway Amazon.

“El almacenamiento en caché bien administrado elimina parcial o completamente algunas interacciones cliente-servidor, mejorando aún más la escalabilidad y el rendimiento.”

Desventaja: Saber controlar las actualizaciones del cache.

RESTRICCIONES REST

4. UNIFORM INTERFACE (Interfaz uniforme Web): **Orientado a recursos**, para la transferencia de datos en un sistema REST utiliza POST, GET, PUT, DELETE (CRUD), este aplica acciones concretas sobre los recursos, siempre y cuando estén identificados con una URI. Los llamados hiperenlaces, **HATEOAS**.

Método HTTP (comando)	Descripción
GET	Solicita el recurso correspondiente al servidor sin modificar su estado
POST	Crea un nuevo recurso subyacente al recurso facilitado y el URI lo dirige automáticamente. También puede usarse para modificar recursos ya existentes
HEAD	Solo solicita el encabezado del recurso al servidor para comprobar, por ejemplo, la validez de un archivo
PUT	Coloca el recurso solicitado en el servidor o modifica uno ya existente
PATCH	Modifica una parte del recurso facilitado
DELETE	Elimina el recurso
TRACE	Devuelve la solicitud tal y como la recibe el servidor web para determinar los cambios en el camino al servidor
OPTIONS	Muestra una lista de los métodos soportados por el servidor
CONNECT	Tramita la solicitud a través de un túnel SSL para, por lo general general, establecer una conexión por medio de un servidor proxy

Client / Request / Headers

POST /services/jwt/issue/adfs HTTP/1.1
Host: federation-xxxxxx.com
Cache-Control: no-cache
Postman-Token: 0d891f9d-9523-67d0-c173-912c52aba117
Content-Type: application/x-www-form-urlencoded

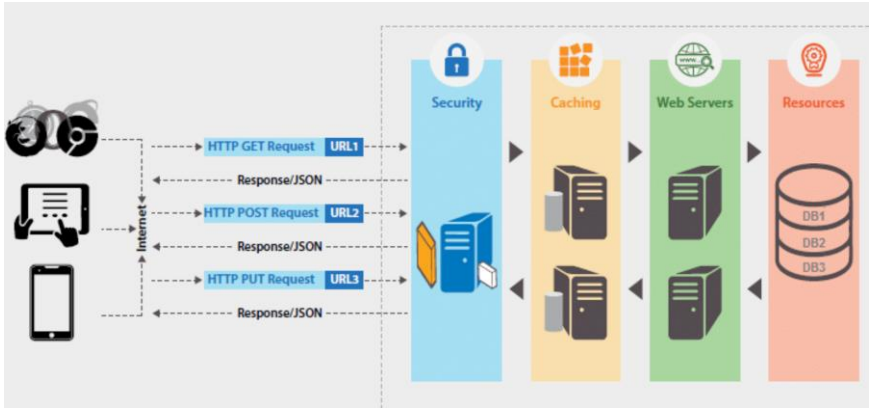
<<JSON REQUEST BODY>>

Server / Response / Headers

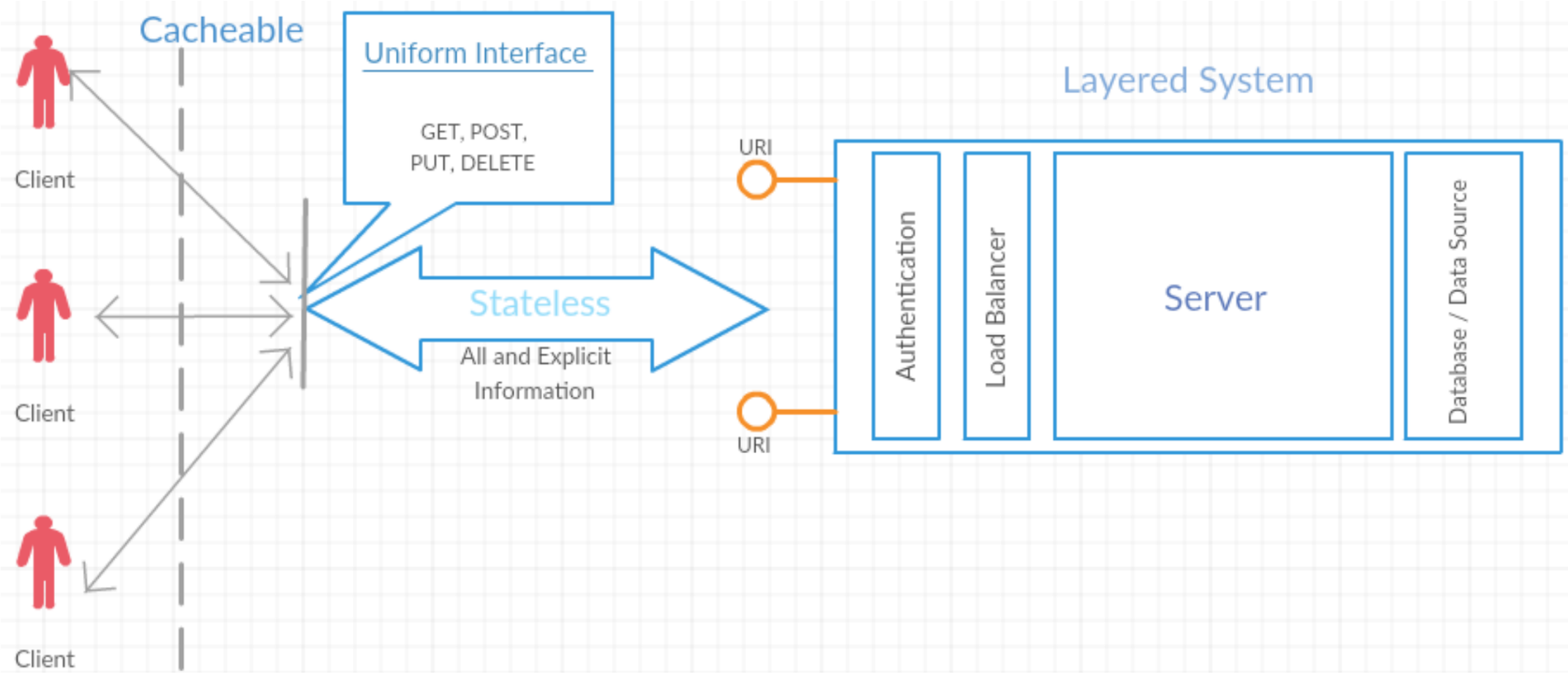
HTTP/1.1 200 OK
Access-Control-Allow-Methods:GET, PUT, POST, DELETE, HEAD, OPTIONS
Access-Control-Allow-Origin:*
Cache-Control:no-cache
Content-Encoding:gzip
Content-Length:1165
Content-Type:application/json; charset=utf-8
Date:Sun, 18 Feb 2018 06:40:14 GMT
Expires:-1
Pragma:no-cache
Server:Microsoft-IIS/10.0
Servername:xxxxxx
Vary:Accept-Encoding,Accept-Encoding
X-AspNet-Version:4.0.30319

RESTRICCIONES REST

5. LAYERED : Sistema de Capas transparente, un cliente REST no será capaz de distinguir entre si esta realizando una petición directamente al servidor, **el servidor puede disponer de varias capas** para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

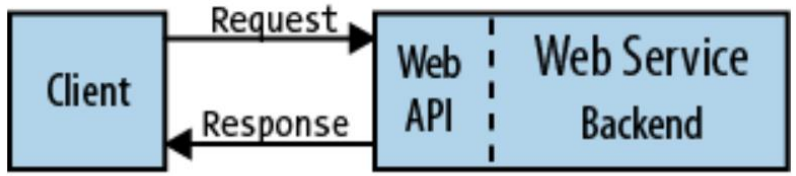


6. CODE ON DEMAND (opcional): Está permitido que el servidor envía código al cliente (después de la implementación) y lo ejecute como un applet o javascript, extensibilidad.


Cacheable
Uniform Interface
GET, POST,
PUT, DELETE
Layered System
Client
Client
Client
Stateless
All and Explicit
Information
URI
URI
Authentication
Load Balancer
Server
Database / Data Source

API REST

Una API web es la *cara (interface)* de un servicio web, escucha y responde directamente a las solicitudes de los clientes.

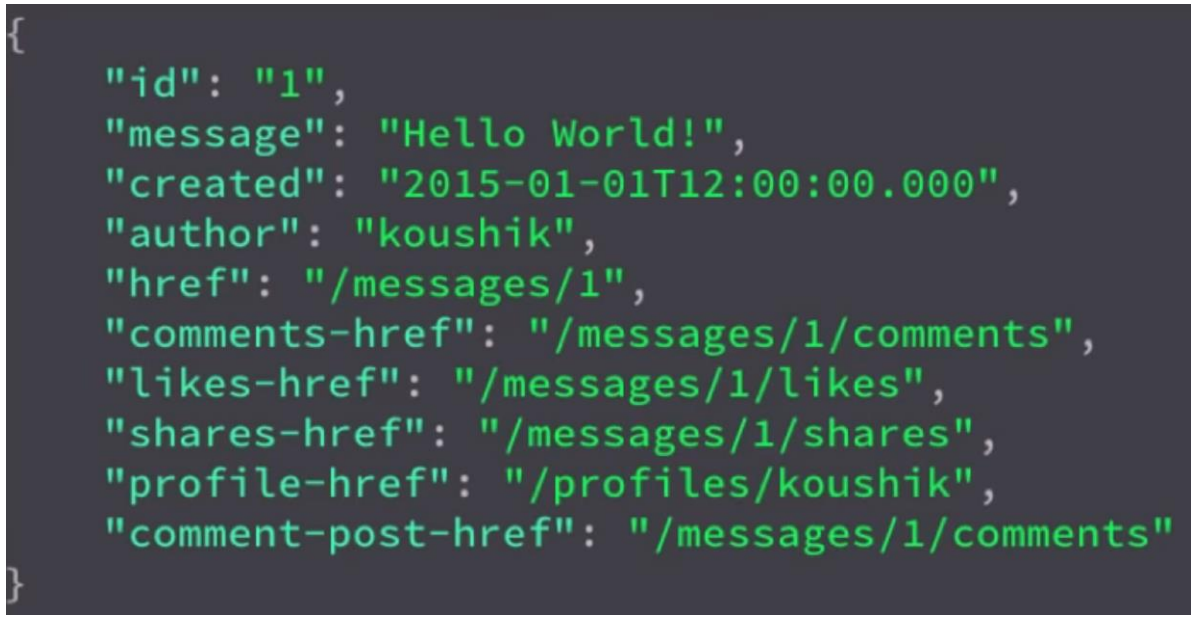


Una API web que se ajusta al estilo arquitectónico REST es una *API REST* . Tener una API REST hace que un servicio web sea "RESTful“.

Una API REST consiste en un conjunto de recursos interconectados. Este conjunto de recursos se conoce como *modelo de recursos de* la API REST .

HATEOAS (Hypermedia As The Engine Of Application State) – USO de Hipermedios – Parte de la restricción REST Interfaz Uniforme

Para cualquier API REST es obligatorio disponer del **principio HATEOAS** para ser una verdadera **API REST.** Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.



HATEOAS significa una representación del estado de la aplicación (recurso) que incluye enlaces a recursos relacionados. La ausencia o presencia de un enlace en una página es una parte esencial del estado actual del recurso y, por lo tanto, es esencial para las API RESTful.

Without HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “owner” : “jim”
}
```

With HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “modify”,
                “href” : “/items/1234”
            },
            {
                “rel” : “delete”,
                “href” : “/items/1234”
            }
        ]
    }
}
```

Control de cache en Cliente

Headers	Descripción	Muestras
Expires	Atributo de Cabecera que representa fecha /tiempo tras el cual la respuesta se considera muerta	Vence: viernes, 12 de enero de 2018 18:00:09 GMT
Cache-control	Un encabezado que define varias directivas (para solicitudes y respuestas) seguidas de mecanismos de almacenamiento en caché	**Max age=4500** extensión de caché
E-Tag	Identificador único para los estados de recursos del servidor	**ETag:uqv2309u324klm**
Last-modified	El encabezado de respuesta ayuda a identificar el momento en que se generó la respuesta	Última modificación: viernes, 12 de enero de 2018 18:00:09 GMT

Algunas APIs REST

https://developers.google.com/youtube/v3/?hl=es-419

http://omdbapi.com/

https://developer.marvel.com/

https://developer.spotify.com/documentation/web-api/

https://currencylayer.com/documentation

https://developer.twitter.com/en/docs

Spring Boot REST

```java
@RestController
@RequestMapping("/api")
public class ProductoController {
   @Autowired
   public Negocio negocio;

   @GetMapping("/productos")
   public List<Producto>  obtenerProductos(){
       return negocio.obtenerProductos();
   }
   @PostMapping("/producto")
   public Producto crearProducto(@Valid @RequestBody Producto producto) {
       Producto p;
       try {
           p = negocio.crearProducto(producto);
       }catch(Exception e){
           throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Mi mensaje", e);
       }
       return p;
   }

   @PutMapping("/producto/{codigo}")
   public Producto actualizarProducto(@PathVariable(value="codigo") Long codigo,
           @Valid @RequestBody Producto productoDetalle) {
     return negocio.actualizarProducto(codigo, productoDetalle);
   }

   @DeleteMapping("/producto/{codigo}")
   public ResponseEntity<String> borrarProducto(@PathVariable(value = "codigo") Long codigo){
      return negocio.borrarProducto(codigo);
   }
```

Control de Excepciones Backend / REST

```java
@GetMapping("/entidad/{codigo}")
public Producto obtenerEntidad(@PathVariable(value = "codigo") Long codigo){
    Producto p;
    try {
        p = negocio.obtenerEntidad(codigo);
    }catch(Exception e){
        System.out.println(e.getMessage());
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Mi mensaje", e);
    }
    return p;
}
```


```json
{
    "timestamp": "2019-06-04T16:11:42.805+0000",
    "status": 404,
    "error": "Not Found",
    "message": "Mi mensaje",
    "path": "/api/entidad/3"
}
```