Tema 7. Servicios web

DESARROLLO EN ENTORNO SERVIDOR 2º DAW

MARÍA CRIADO DOMÍNGUEZ

Índice

- Servicio web
- Características
- Tipos
- Api REST
  - Generar servicio
  - Consumir servicio

Servicio web

Utilizamos los servicios web cuando las aplicaciones que desarrollamos tienen que compartir información con otras aplicaciones web.

- Queremos compartir esta información sin dar acceso a la misma base de datos a dos o más aplicaciones. Solo una aplicación accede a la base de datos.
- Queremos compartir una lógica de negocio que hemos programado en una aplicación para que pueda ser utilizada (sin necesidad de volver a codificarla) en otra aplicación. Reutilización de código desarrollado en otra aplicación.
- Queremos controlar las modificaciones que otras aplicaciones realizan sobre nuestra base de datos. Validación de las modificaciones en nuestros datos.

Servicio web

Partes

- El servidor puede ofrecer un punto de acceso a la información que quiere compartir. Controla y facilita el acceso a otras aplicaciones.
- Los clientes no necesitan conocer la estructura de almacenamiento, únicamente el punto de acceso a la información que les interesa.

Alternativas

- SOAP. Protocolo que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML (Cliente-Servidor)
- WSDL. Web Services Description Language, es un formato del Extensible Markup Language (XML) que se utiliza para describir servicios web
- REST. La Transferencia de Estado Representacional (Representational State Transfer) o REST es una arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, POST, PUT, DELETE, etcétera) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

API

Es una forma de describir la manera en que los programas o los sitios webs intercambian datos.

El formato de intercambio de datos normalmente es **JSON** o XML.

Una API especifica cómo los componentes de software deben interactuar entre sí.

Es un conjunto de protocolos y rutinas, y sus respuestas generalmente se devuelven como datos JSON o XML.

API

El cliente no necesita saber qué procedimiento se llama en el servidor.

En su lugar, utiliza un conjunto de comandos (llamados “verbos”) que están integrados en HTTP y cuando el comando llega al otro extremo, depende del sistema receptor saber qué hacer con él.

Por ejemplo, el verbo HTTP que se usa para recuperar datos es “GET”.

El sistema cliente (generalmente llamado “*consumidor*”) y el sistema servidor (el “*proveedor*”) son tan independientes entre sí, que pueden usar diferentes lenguajes (***Java, Python, Ruby, PHP, etc.***) para su implementación general.

Arquitectura REST

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Esto es, nada de variables de sesión en el servidor.

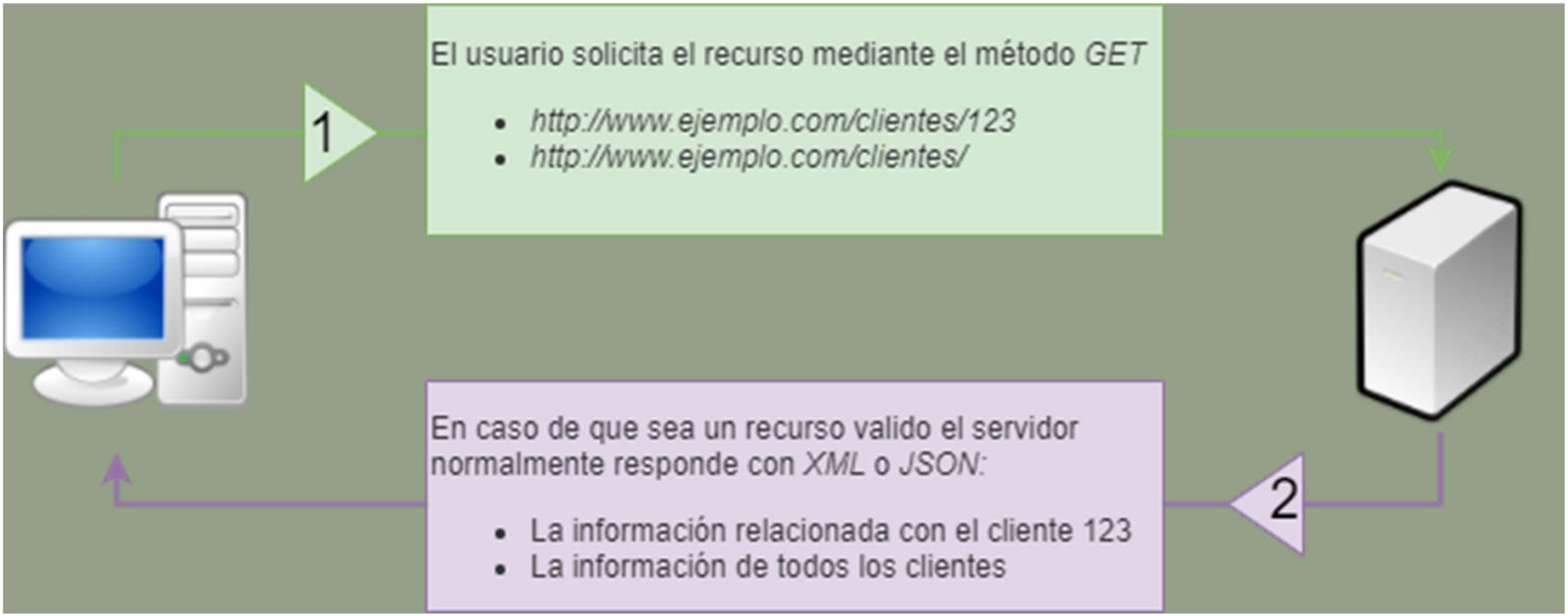
- REST pide a los desarrolladores que utilicen el protocolo HTTP de forma explícita y de una forma que sea coherente con la definición del protocolo.

- Este principio básico del diseño de REST establece una correlación individual entre las operaciones de crear, leer, actualizar y borrar (CRUD) y los métodos HTTP.

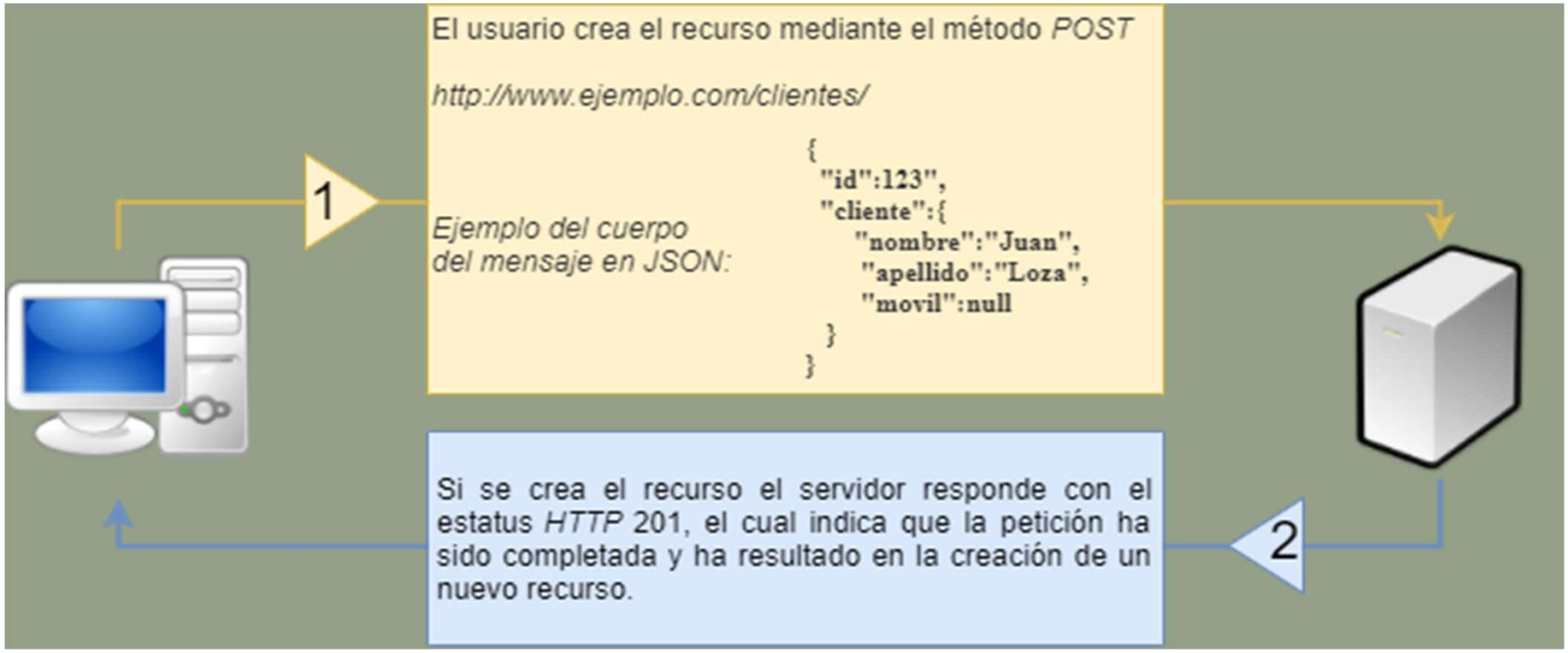
Arquitectura REST

Operación	Método HTTP	URI	Parámetros	Resultado
Listar	GET	/{recurso}	No aplica	Lista del tipo de recurso, archivo JSON
Crear	POST	/{recurso}	Archivo JSON	Se crea un nuevo recurso
Leer	GET	/{recurso}/{recurso_id}	No aplica	Recurso en función al id
Actualizar	PUT	/{recurso}/{recurso_id}	Archivo JSON	Se actualiza el recurso
Borrar	DELETE	/{recurso}/{recurso_id}	No aplica	Se elimina el recurso en función al id

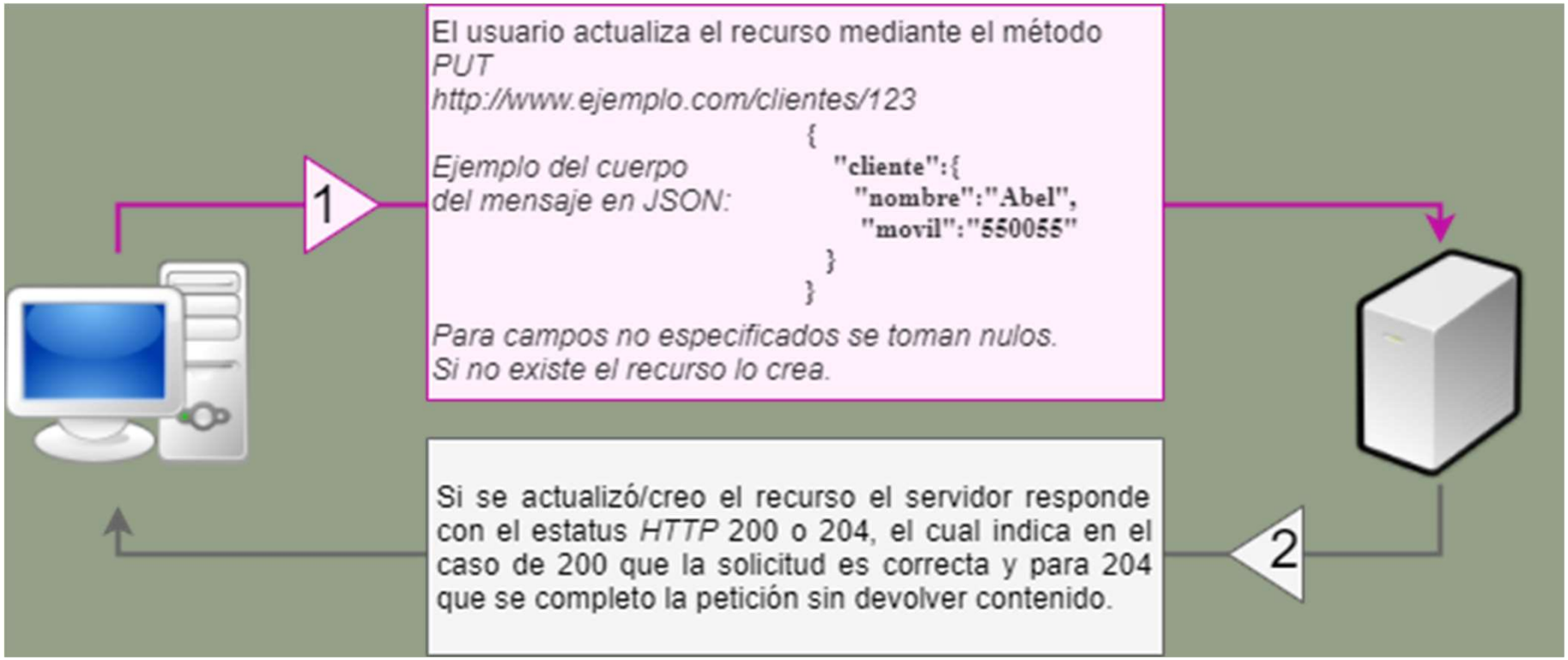
Arquitectura REST
GET



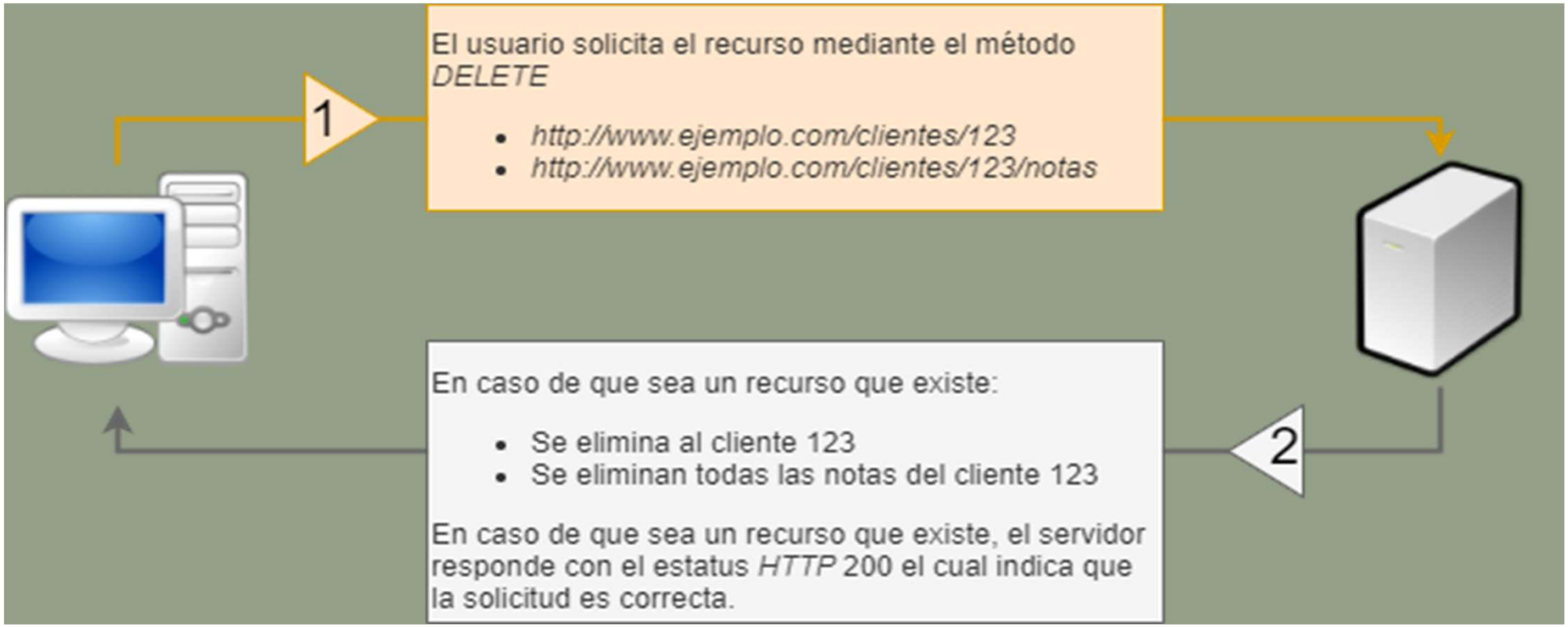
Arquitectura REST
POST



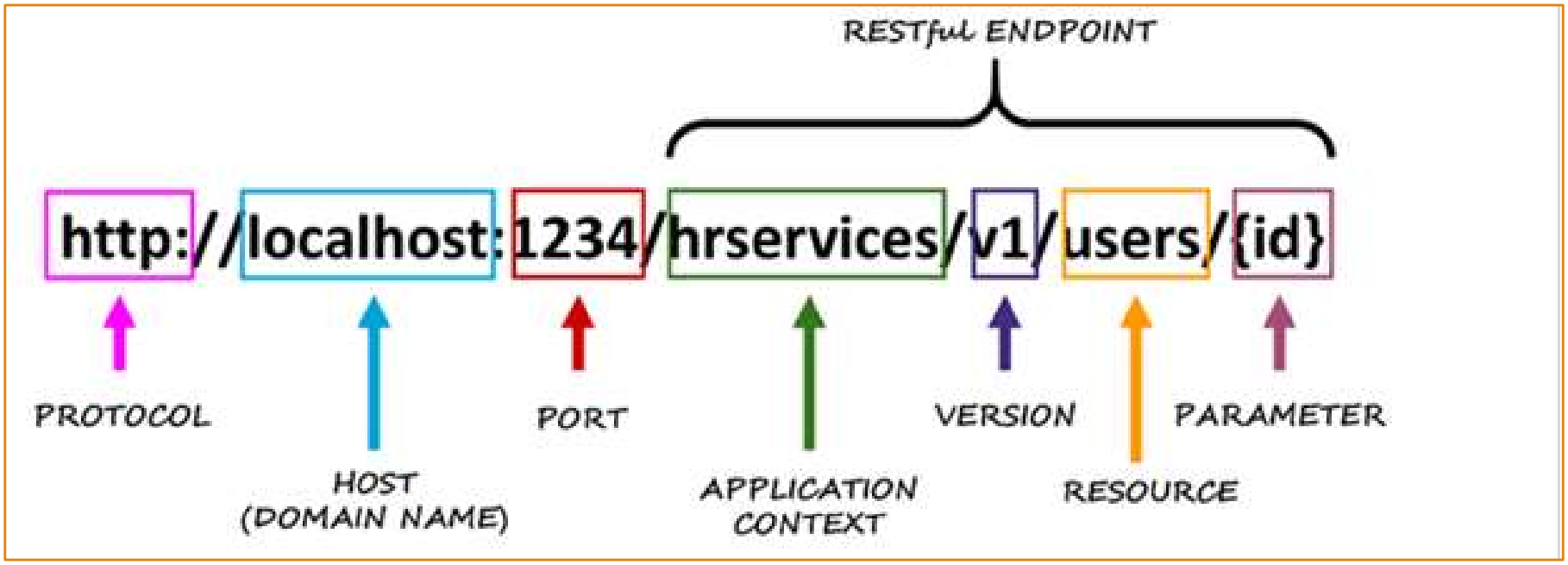
Arquitectura REST
PUT



Arquitectura REST
DELETE



Arquitectura REST
Endpoint



Arquitectura REST
Endpoint

Existen varias reglas básicas para ponerle nombre a la URI de un recurso:

- Los nombres de URI no deben implicar una acción, por lo tanto debe evitarse usar **verbos** en ellos.
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independiente de formato.
- Deben mantener una jerarquía lógica.
- Los **filtrados de información de un recurso no se hacen en la URI**, sino con una cadena de consulta (string query).

Arquitectura REST
Endpoint

Existen varias reglas básicas para ponerle nombre a la URI de un recurso:

- Los nombres de URI no deben implicar una acción, por lo tanto debe evitarse usar **verbos** en ellos.
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independiente de formato.
- Deben mantener una jerarquía lógica.
- Los **filtrados de información de un recurso no se hacen en la URI**, sino con una cadena de consulta (string query).

Arquitectura REST
Endpoint

- La URI /facturas/orden/desc/fecha-desde/2007/pagina/2 sería incorrecta ya que el recurso de listado de facturas sería el mismo pero utilizaríamos una URI distinta para filtrarlo, ordenarlo o paginarlo.

- La URI correcta en este caso sería:

  /facturas?fecha-desde=2007&orden=DESC&pagina=2

Arquitectura REST
Resultado

RESTful exige que sea enviado un código de estado HTTP informando del resultado de la petición.

Para enviar la repuesta en PHP debemos utilizar un header, con el código correspondiente, ejemplo:

Header ("HTTP/1.1 **200 OK**");

- 201 Recurso creado correctamente
- 400 Petición errónea. Esto puede estar causado por varias acciones del usuario, como proveer un JSON no válido en el cuerpo de la petición, proveyendo parámetros de acción no válidos, etc.
- 404 No encontrado El recurso solicitado no existe.
- 405 Método no permitido El método HTTP de la solicitud no se permite en el recurso.
- 406 Not Acceptable. Indica que el servidor no puede producir una respuesta que coincida con la lista de valores aceptables definidos en las cabeceras del cliente, por ejemplo, pide xml y el recurso genera JSON.
- 500 Error interno del servidor Se produjo un error interno del servidor al procesar la solicitud. Por ejemplo, al conectar con la Base de Datos.

CORS

CORS son las siglas de "Cross-origin resource sharing" y es básicamente una restricción de acceso a recursos que están localizados en otros dominios.

Tu página o aplicación puede estar en http://midominio.com y tu servidor de API (PHP) puede estar en http://otrodominio.com

Para evitar que la barrera de CORS nos afecte, el servidor tiene que emitir unas cabeceras HTTP en la respuesta.

Estas cabeceras le dicen al navegador que ciertos recursos sí que van a estar disponibles desde otros dominios distintos del habitual.

Así que desde PHP es tan sencillo como escribir unas pocas líneas de código para el envío de esas cabeceras y nuestra API estará disponible para el acceso desde otros dominios.

Las cabeceras son las siguientes.

```
header('Access-Control-Allow-Origin: *');

header("Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept");

header('Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE');
```