

Arquitecturas de Software para Aplicaciones Empresariales

Introducción a los servicios RESTFul

PROGRAMA DE INGENIERIA DE SISTEMAS

Ing. Daniel Eduardo Paz Perafán (danielp@Unicauca.edu.co)

Ing. Pablo A. Magé (pmage@Unicauca.edu.co)



- ❖ Conceptos generales
 - Servicio
 - Interface de programación de aplicaciones
 - Componentes de la web
 - Servicio web RESFul
- ❖ Elementos del protocolo HTTP
- ❖ Servicios RESTFul

❖ Conceptos generales

- Servicio
- Interface de programación de aplicaciones
- Componentes de la web
- Servicio web RESFul

❖ Elementos del protocolo HTTP

❖ Servicios RESTFul

- ❖ Un servicio es una unidad de software, constituida por una o más funciones que realizan una tarea específica
- ❖ Contiene las integraciones de datos y código que se necesitan para llevar a cabo una tarea empresarial completa y diferenciada.
- ❖ El **World Wide Web Consortium (W3C)** define un servicio como un sistema de software designado para dar soporte a la interoperabilidad de máquina a máquina a través de una red.

La URL de la petición es `localhost:8085/api/clientes/`

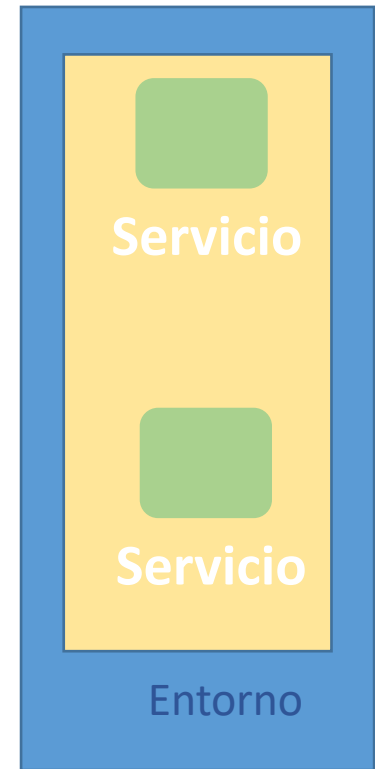
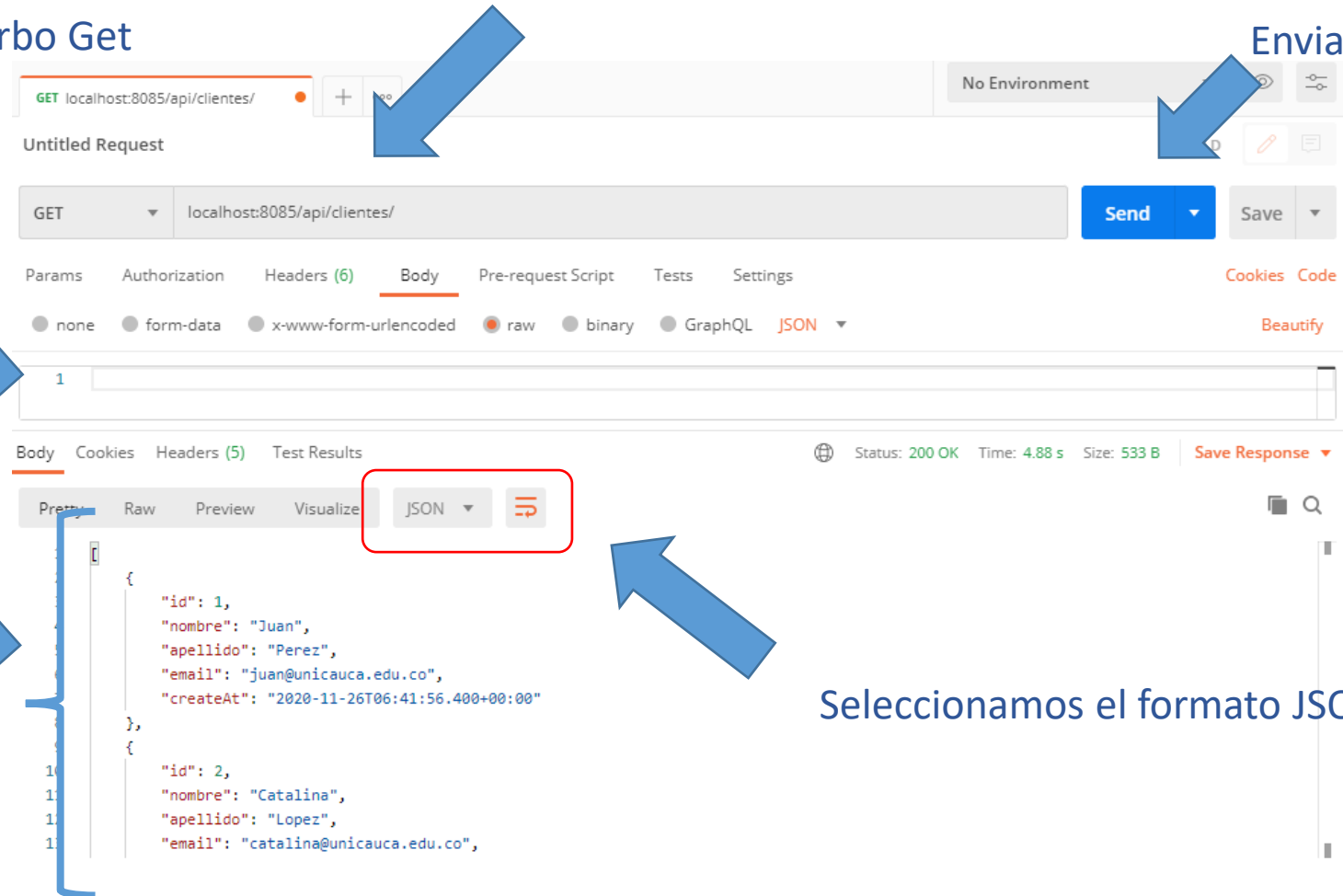
Seleccionamos el verbo Get

Enviamos la petición

No enviamos nada en el cuerpo de la petición

Se listan los clientes en formato JSON que el servicio web retorna

Seleccionamos el formato JSON



Maquina

Ejemplos de servicios

SIMCA podría ofrecer los siguientes servicios:

- Determinar si un estudiante esta activo en la Universidad del Cauca
- Consultar los datos de un estudiante de la Universidad del Cauca

El área de Personal docente podría ofrecer los siguientes servicios:

- Listar los docentes activos en la Universidad del Cauca.
- Consultar las asignaturas dictadas por un docente

El área de bibliotecas podría ofrecer los siguientes servicios

- Consultar si un estudiante tiene libros por devolver.
- Consultar el promedio de libros solicitados por los estudiantes de un determinado programa

Los servicios pueden ser contruidos en diferentes lenguajes de programación y utilizando diferentes tecnologías.

Los servicios utilizan **protocolos, formatos, modelos y tecnologías propias de los lenguajes** para su construcción.

- Lenguajes de programación: C, C++, java, Python, typeScript
- Protocolos como SOAP, HTTP, TCP, IP
- Formatos como JSON, XML
- Modelos como REST

Los principales servicios son los WEB, los cuales se basan en los elementos de la WEB.

Ejemplos de servicios

Universidad del Cauca



Roles de la
organización

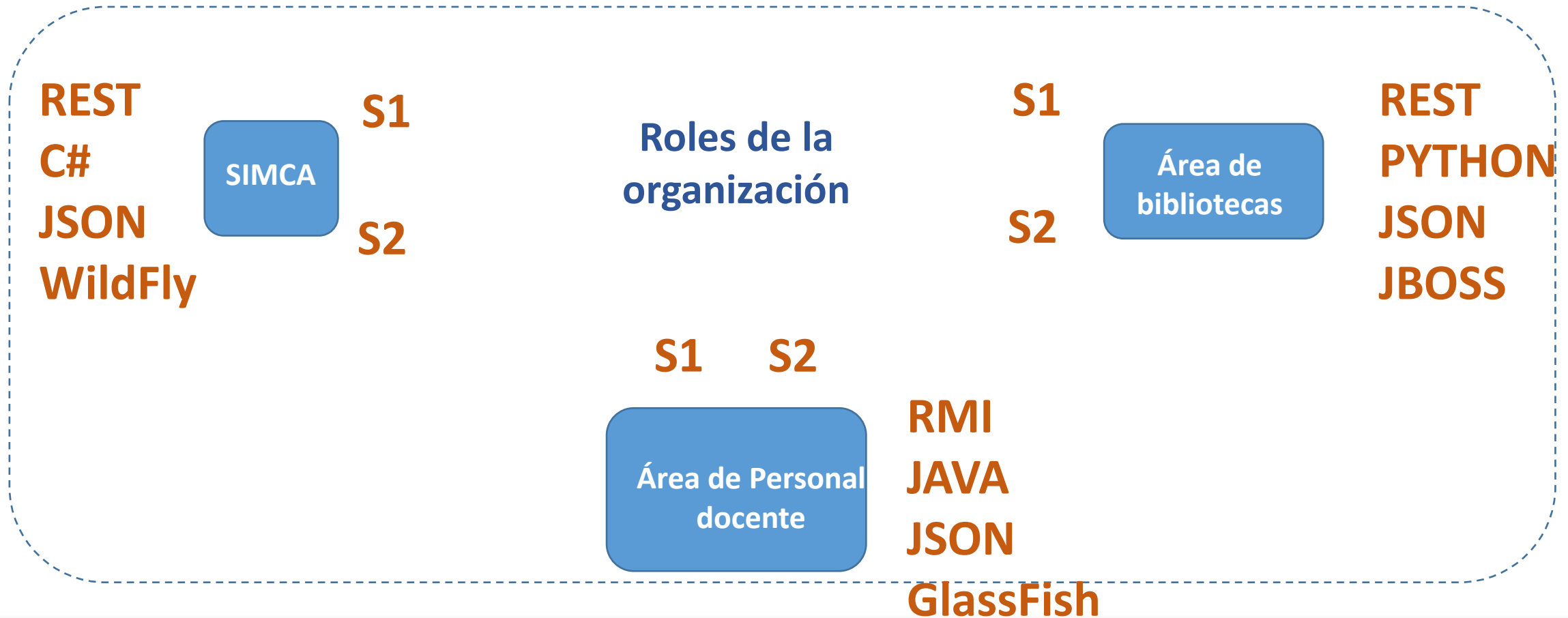
S1
S2



S1 S2



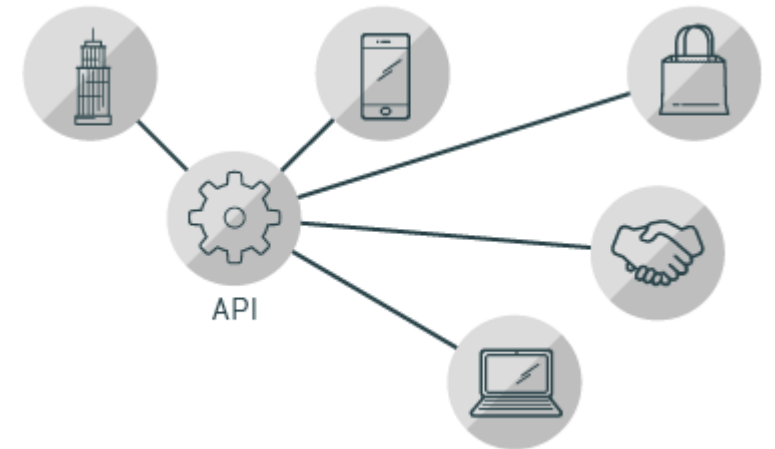
Universidad del Cauca



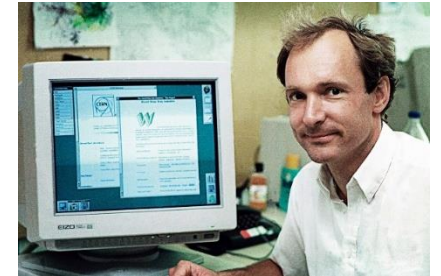
Interfaz de programación de aplicaciones (API) es un conjunto de herramientas, definiciones y protocolos que se utiliza para integrar los servicios y el software de aplicaciones

Las API pueden ser:

- Privadas (para uso interno únicamente)
- Compartidas (con partners específicos para brindar flujos de ingresos adicionales)
- Públicas (entidades externas pueden desarrollar aplicaciones que interactúen con sus API para fomentar la innovación).



- ❖ En 1989, el físico Tim Berners-Lee llevaba varios años trabajando en el CERN, la Organización Europea para la Investigación Nuclear.



- ❖ Los científicos del CERN eran cientos que generaban informes, documentos, y diseños, pero no podían compartir la información que generaban
- ❖ En marzo de 1989, Tim escribió un pequeño informe, en el que proponía el desarrollo de un sistema distribuido, la www, para almacenar y compartir la información.
- ❖ El sistema planteado es un medio para la distribución de la información entre equipos de investigadores geográficamente dispersos.
- ❖ La primera página web se publicó a finales de 1990.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Example</title>
5     <link rel="stylesheet" href="sty"
6   </head>
7   <body>
8     <h1>
9       <a href="/">Header</a>
10    </h1>
11    <nav>
12      <a href="/one/">One</a>
13      <a href="/two/">Two</a>
14      <a href="/three/">Three</a>
15    </nav>
```

No invento nos **hiper-enlaces**. El concepto lo utilizó en su modelo de sistema distribuido.

Las **innovaciones** que Berners-Lee desarrolló para dar forma a su idea fueron tres:

HTML (*hypertext markup language*), es el lenguaje que permite estructurar los documentos web

URL (*uniform resource location*), que son las direcciones que permiten encontrar los documentos web

HTTP (*hypertext transfer protocol*), protocolo que permite la comunicación entre un cliente y un servidor, creado para transmitir los documentos web

Un cliente que muestre (e incluso pueda editar) esos documentos. El primer navegador Web, llamado: WorldWideWeb.

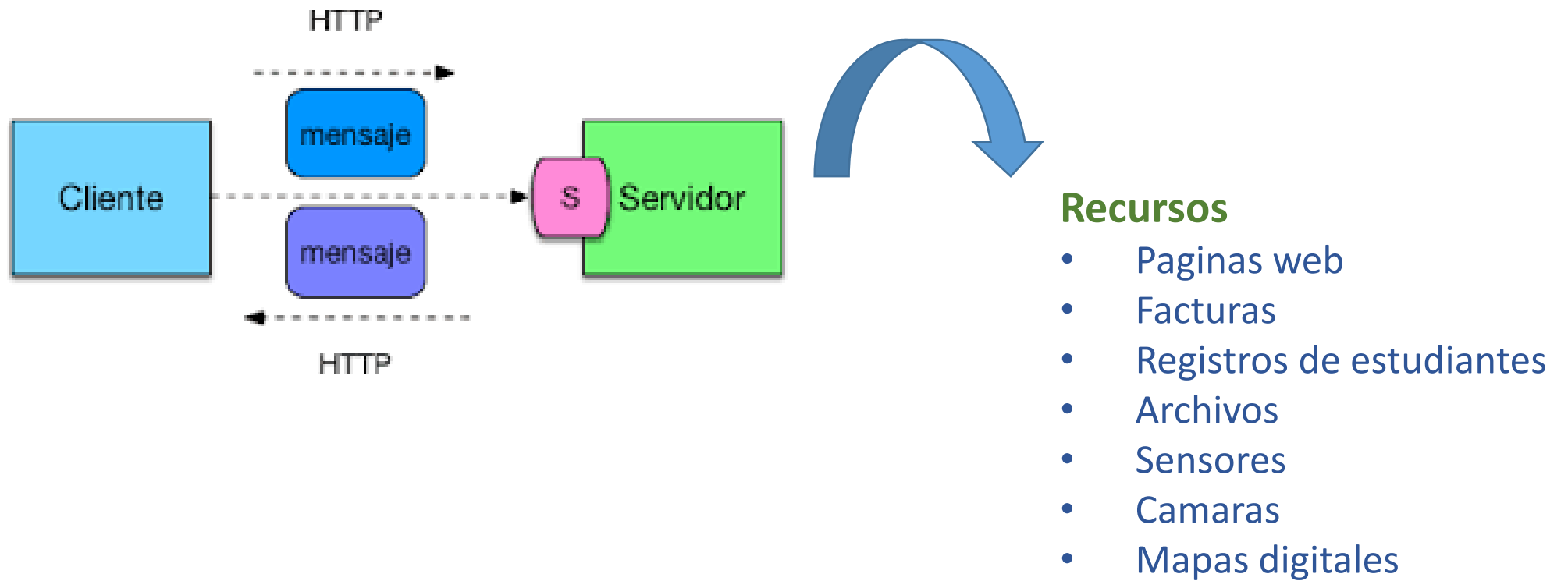


Tim Berners-Lee vende un NFT del primer código fuente de la web por 5,4 millones de dólares

“Esto está totalmente alineado con los valores de la web”, ha expresado sobre los NFT el padre la tecnología World Wide Web, que donará el dinero a causas benéficas

- Tim Berners-Lee ha vendido:
- Una carta donde expresa porque vende el código fuente
- Las 9.555 líneas de código original de la World Wide Web
- Un vídeo de unos 30 minutos que las muestra como si se estuvieran escribiendo en ese momento
- Un poster digital por 5,4 millones de dólares.

Un **Servicio Web** es una API que se comunica mediante HTTP y URLs



- **REST** (Representational State Transfer o Transferencia de Estado Representacional) es un modelo que define lineamientos sobre como deben enviarse y recibirse los mensajes sobre HTTP.

Dado que se trata de un conjunto de pautas, la implementación de las recomendaciones depende de los desarrolladores.

- Específicamente definen la manera en que se diseñan las interfaces de programación de aplicaciones (API)
- **RESTful** hace referencia a un servicio web que se implementa siguiendo los lineamientos del modelo REST.

❖ Conceptos generales

- Servicio
- Interface de programación de aplicaciones
- Servicio web RESFul
- Componentes de la web

❖ Elementos del protocolo HTTP

❖ Servicios RESTFul

Para desarrollar APIs REST los aspectos claves que hay que dominar y tener claros son:

- Partes HTTP
- Métodos HTTP
- Aceptación de tipos de contenido
- Códigos de estado

HTTP es el **protocolo** que permite enviar documentos de un lado a otro en la web.

Un **protocolo** define las reglas y la estructura de los mensajes que se van a intercambiar entre máquinas.

Un mensaje **HTTP** (no importa si es de petición o respuesta) se compone de 3 partes:

- La primera línea (que es diferente para la petición y la respuesta).
- Los encabezados.
- El cuerpo (opcional)

Invencción de la World Wide Web

HTTP/0.9 – El protocolo de una sola línea

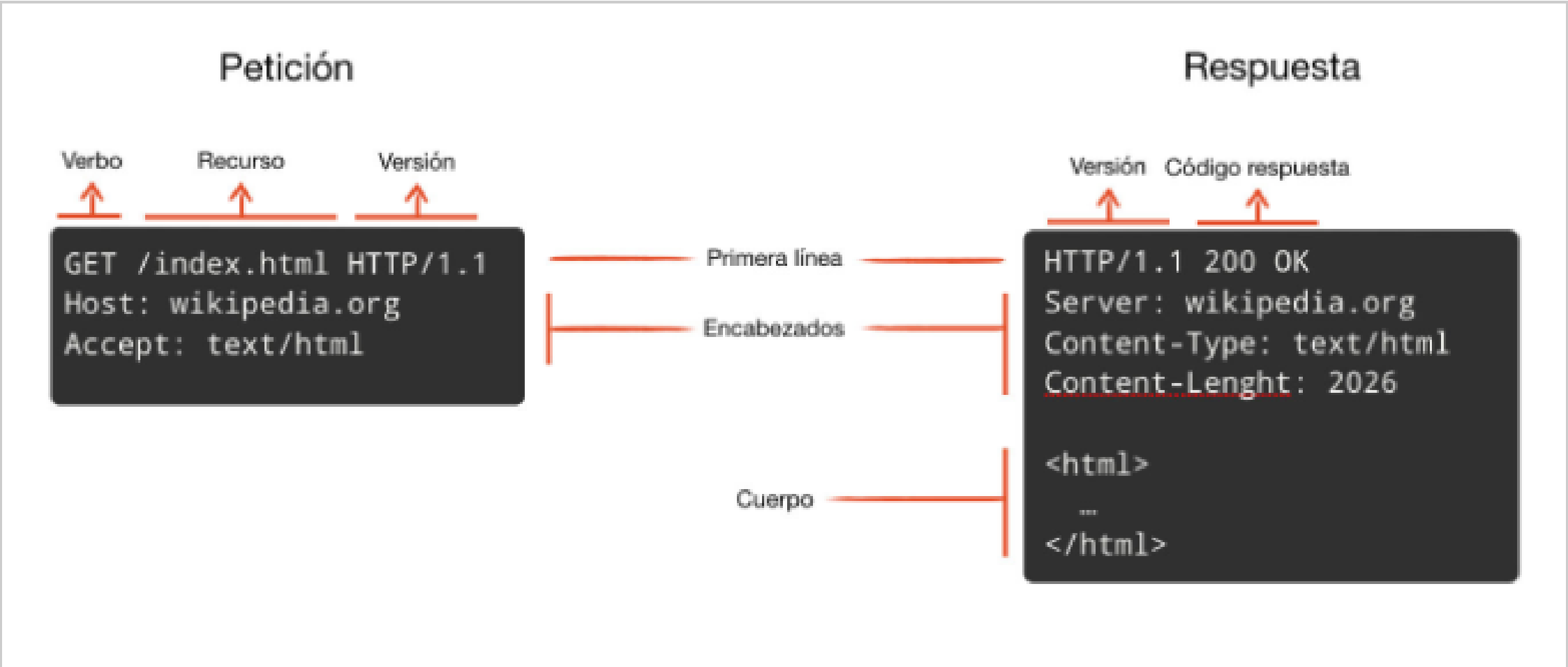
HTTP/1.0 – Desarrollando expansibilidad

HTTP/1.1 – El protocolo estándar.

Más de 15 años de expansiones

HTTP/2 – Un protocolo para un mayor rendimiento

Post-evolución del HTTP/2



Ejemplo de un mensaje de petición (sin cuerpo):

```
GET /index.html HTTP/1.1
Host: wikipedia.org
Accept: text/html
```



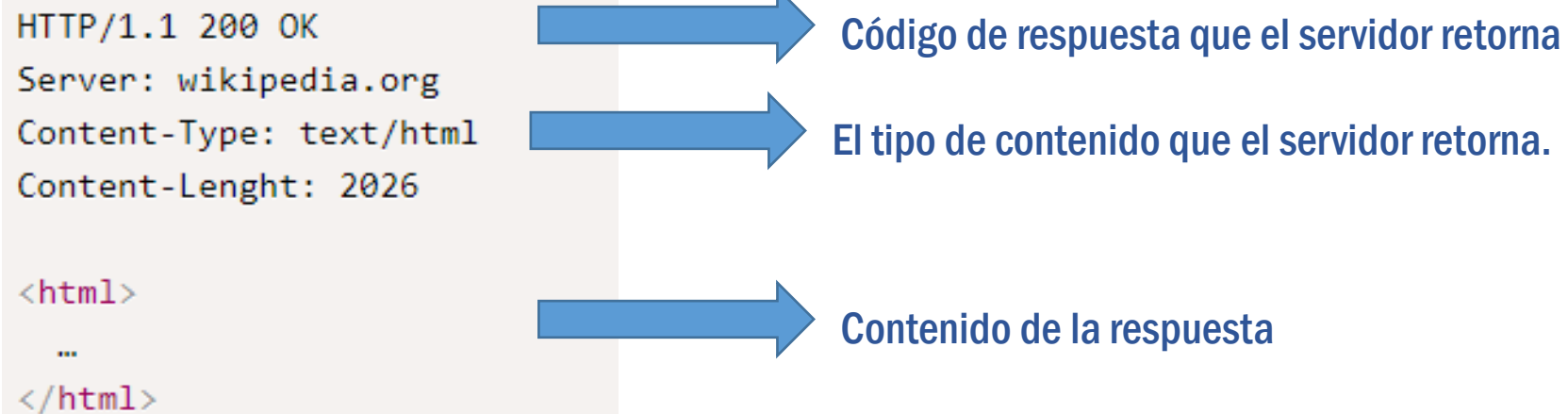
Métodos a ejecutar y recurso solicitado.

El tipo de contenido que el cliente está esperando.

Ejemplo de un mensaje de respuesta:

```
HTTP/1.1 200 OK
Server: wikipedia.org
Content-Type: text/html
Content-Length: 2026

<html>
...
</html>
```



Código de respuesta que el servidor retorna

El tipo de contenido que el servidor retorna.

Contenido de la respuesta

Ejemplo de un mensaje de petición (sin cuerpo):

```
GET /index.html HTTP/1.1  
Host: wikipedia.org  
Accept: text/html
```



Las cabeceras (en inglés headers) HTTP permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta.

Ejemplo de un mensaje de respuesta:

```
HTTP/1.1 200 OK  
Server: wikipedia.org  
Content-Type: text/html  
Content-Length: 2026
```

```
<html>  
...  
</html>
```



Una cabecera de petición esta compuesta por su nombre (no sensible a las mayúsculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea).

Ejemplo de un mensaje de petición (sin cuerpo):

```
GET /index.html HTTP/1.1
Host: wikipedia.org
Accept: text/html
```



La cabecera **Accept** anuncia que tipo de contenido el cliente puede procesar, expresado como un tipo MIME

Ejemplo de un mensaje de respuesta:

```
HTTP/1.1 200 OK
Server: wikipedia.org
Content-Type: text/html
Content-Lenght: 2026
```



```
<html>
...
</html>
```

Un **tipo de medio** (también conocido como **extensiones de correo de Internet multipropósito o tipo MIME**) indica la naturaleza y el formato de un documento, archivo o variedad de bytes.

La cabecera **Content-Type** dice al cliente que tipo de contenido será retornado

Partes de HTTP

La URL de la petición es `localhost:8085/api/clientes/`

Seleccionamos el verbo Get

Enviamos la petición

No enviamos nada en el cuerpo de la petición

Se listan los clientes en formato JSON que el servicio web retorna

Seleccionamos el formato JSON

The screenshot shows a REST client interface with the following components:

- Request Bar:** Shows the method `GET` and the URL `localhost:8085/api/clientes/`. A blue arrow points to the URL.
- Send Button:** A blue button labeled `Send`. A blue arrow points to it.
- Body Tab:** The `Body` tab is selected. The body is empty. A blue arrow points to the body area.
- Response Tab:** The `Response` tab is selected. It shows the status `200 OK`, time `4.88 s`, and size `533 B`. A blue arrow points to the response area.
- JSON Format:** The response is displayed in `JSON` format. A red box highlights the `JSON` dropdown and the `Beautify` icon. A blue arrow points to this area.
- Response Data:** The response is a JSON array of two client objects:

```
{
  "id": 1,
  "nombre": "Juan",
  "apellido": "Perez",
  "email": "juan@unicauca.edu.co",
  "createAt": "2020-11-26T06:41:56.400+00:00"
},
{
  "id": 2,
  "nombre": "Catalina",
  "apellido": "Lopez",
  "email": "catalina@unicauca.edu.co",
  "createAt": "2020-11-26T06:41:56.400+00:00"
}
```

La primera línea de un mensaje de petición empieza con un **verbo** (también se le conoce como **método**). Los **verbos** definen la acción que se quiere realizar sobre el recurso.

- **GET:** Para consultar y leer recursos
- **POST:** Para crear recursos
- **PUT:** Para editar recursos
- **DELETE:** Para eliminar recursos.
- **PATCH:** Para editar partes concretas de un recurso.
- **HEAD:** Este método se utilizar para obtener información sobre un determinado recurso sin retornar el registro.

Ejemplos

- **GET /facturas** Nos permite acceder al listado de facturas (Las facturas serán un recurso)
- **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

Las cabeceras (en inglés *headers*) HTTP permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta.

Una cabecera de petición esta compuesta por su nombre (no sensible a las mayúsculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea)

Cabecera Host

El encabezado de solicitud Host especifica el nombre de dominio del servidor (para hosting virtual), y (opcionalmente) el número de puerto TCP en el que el servidor esta escuchando.

Host: developer.mozilla.org

Cabecera Accept

Anuncia que tipo de contenido el cliente puede procesar, expresado como un tipo MIME. Usando negociación de contenido, el servidor selecciona una de las propuestas, la utiliza e informa al cliente de la elección a través de la cabecera de respuesta Content-Type .

Accept: text/html

Accept: image/*

Accept: text/html, application/xhtml+xml, application/xml

Cabecera Content-Type

Usada para indicar el media type (en-US) del recurso. dice al cliente que tipo de contenido será retornado.

Content-Type: text/html; charset=utf-8

Content-Type: image/png

Content-Type: multipart/form-data; boundary=something

Tipos MIME

El tipo Extensiones multipropósito de Correo de Internet (MIME) es una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos.

Puede encontrar la lista más actualizada y completa en la página [Media Types \(iana.org\)](http://media-types.iana.org)

Los navegadores a menudo usan el tipo MIME (y no la extensión de archivo) para determinar cómo procesará un documento.

Estructura general tipo/subtipo

text/plain

text/html

image/jpeg

image/png

audio/mpeg

audio/*

video/mp4

Tipo	Descripción	Ejemplo de subtipos típicos
text	Representa cualquier documento que contenga texto y es teóricamente legible por humanos	text/plain, text/html, text/css, text/javascript
image	Representa cualquier tipo de imagen. Los videos no están incluidos, aunque las imágenes animadas (como el gif animado) se describen con un tipo de imagen.	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	Representa cualquier tipo de archivos de audio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	Representa cualquier tipo de archivos de video	video/webm, video/ogg
application	Representa cualquier tipo de datos binarios.	application/octet-stream, application/pkcs12, application/vnd.mspowerpoint, application/xhtml+xml, application/xml, application/pdf

La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta.

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

- **1XX**: Información
- **2XX**: Éxito
- **3XX**: Redirección
- **4XX**: Error en el cliente
- **5XX**: Error en el servidor

Estás familiarizado(a) con el famoso error **404** que retornan los servidores cuando el recurso no fue encontrado. O con el error **500** cuando ocurre un error en el servidor. Pero existen muchos más.

201 Created

El código de respuesta de estado de éxito creado HTTP 201 indica que la solicitud ha tenido éxito y ha llevado a la creación de un recurso. El nuevo recurso se crea efectivamente antes de enviar esta respuesta. y el nuevo recurso se devuelve en el cuerpo del mensaje

404 Not Found

El código de error HTTP 404 Not Found (404 No Encontrado) de respuesta de cliente indica que el servidor no puede encontrar el recurso solicitado.

Un código de estado 404 no indica si el recurso está temporalmente o permanentemente ausente. Pero si un recurso es permanentemente eliminado, un 410 (en-US) (Gone) debe ser usado en lugar del estado 404.

❖ Conceptos generales

- Servicio
- Interface de programación de aplicaciones
- Servicio web RESFul
- Componentes de la web

❖ Elementos del protocolo HTTP

❖ Servicios RESTFul

- ❖ Todo lo que se mueve a través de las comunicaciones web es un recurso.
- ❖ Cada uno de estos recursos debe tener un identificador único, el cual va a estar dado por su URL
- ❖ Debe utilizar los verbos estándares de HTTP, que están definidos en el protocolo nativo, donde cada uno de estos verbos significa una acción diferente. Hay definidas 8 acciones principales: GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, CONNECT.
- ❖ Se trata de comunicaciones que se denominan sin estado (STATELESS), es decir, no hay necesidad de que los servicios guarden las sesiones de los usuarios

El objetivo de una solicitud HTTP se denomina "recurso", (es decir: datos), y dicho recurso, no posee un tipo definido por defecto.

Puede ser un registro de una base de datos, un documento, o una foto, o cualquier otra posibilidad.

Cada recurso es identificado por un Identificador Uniforme de Recursos (URI)

La forma más común de URI es la (URL) (de las siglas en ingles: "*Uniform Resource Locator*",

`http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2`

Un URN es una URI que identifica un recurso por su nombre en un espacio de nombres particular.

`urn:isbn:9780141036144` El libro "1984" por George Orwell,

Las URL, **Uniform Resource Locator** permiten **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}

- ❖ En un sistema REST, la **URI** no debe cambiar a lo largo del tiempo.
- ❖ Los **URIs** de los servicios web de REST deberían ser intuitivos
 - La estructura de un URI debería ser bastante clara, predecible y fácil de entender.
 - `http://www.myservice.org/discussion/topics/{topic}`
- ❖ Las **URIs** deben ser independientes de formato

La URI `/facturas/234.pdf` no sería una URI correcta



`/facturas/234`

Un recurso REST es cualquier cosa que sea direccionable a través de la Web.

Algunos ejemplos de **recursos REST** son:

- Una noticia de un periódico
- La temperatura de Alicante a las 4:00pm
- Un valor de IVA almacenado en una base de datos
- Una lista con el historial de las revisiones de código en un sistema CVS
- Un estudiante en alguna aula de alguna universidad
- El resultado de una búsqueda de un ítem particular en Google

Un recurso se identifica con una URL por ejemplo myDomino.com/facturas/234

REST utiliza los **métodos HTTP** de manera explícita, siguiendo el protocolo definido por RFC 2616

Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP

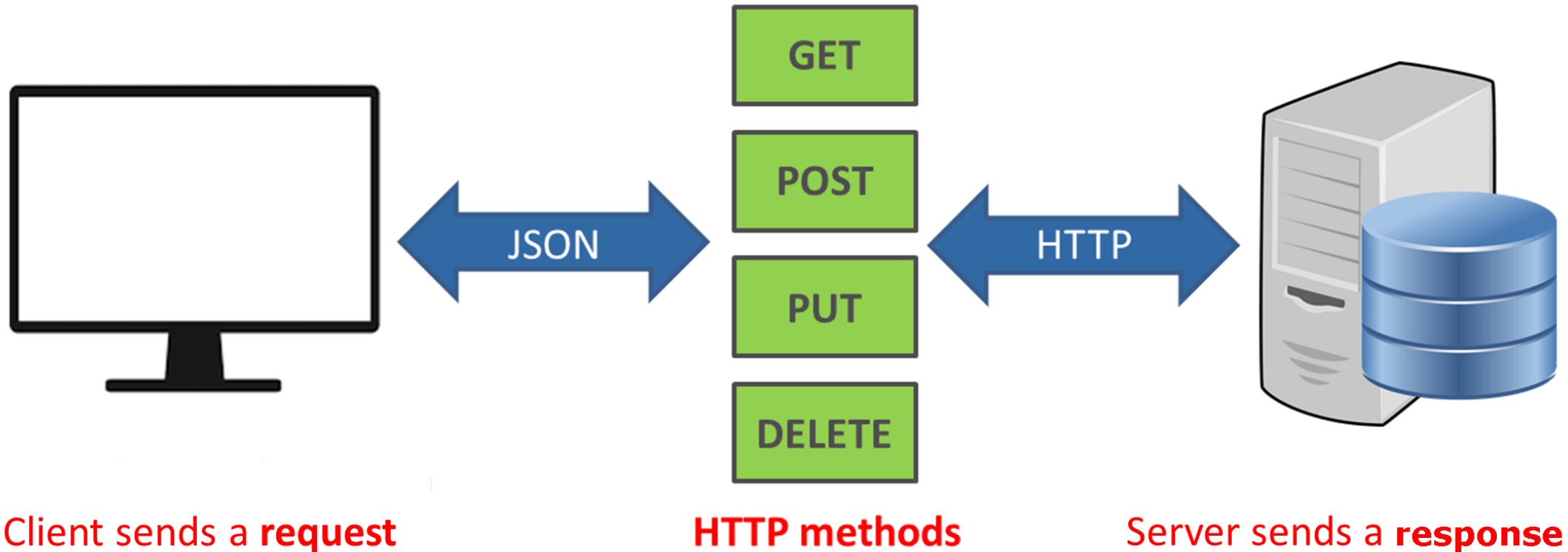
- Para crear un recurso en el servidor hay que utilizar un **POST**.
- Para recuperar un recurso hay que utilizar un **GET**.
- Para cambiar el estado de un recurso, o para actualizarlo, hay que utilizar un **PUT**.
- Para eliminar o borrar un recurso hay que utilizar un **DELETE**.
- Para editar partes concretas de un recurso hay que utilizar **PATCH**:

Una falla de diseño poco afortunada que tienen muchos servicios web es el uso de métodos HTTP para otros propósitos.

- El método HTTP GET para ejecutar algo transaccional en el servidor; por ejemplo, agregar registros a una base de datos.

Operaciones HTTP para ejecutar operaciones

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS



- ❖ Para filtrar, ordenar, paginar o buscar información en un recurso, debemos hacer una consulta sobre la **URI**, utilizando parámetros HTTP en lugar de incluirlos en la misma.

La URI `/facturas/orden/desc/fecha-desde/2007/pagina/2` no es correcta

La URI `/facturas?fecha-desde=2007&orden=DESC&pagina=2` sería correcta

- ❖ Las **URIs** no deben implicar acciones y deben ser únicas

La URI `/facturas/234/editar` sería incorrecta ya que tenemos el verbo editar en la misma.

- ❖ HTTP nos permite especificar en qué formato el cliente desea recibir el recurso, logrando indicar varios en orden de preferencia, para ello utilizamos el campo accept del encabezado.
- ❖ Nuestra API devolverá el recurso en el primer formato disponible y, de no poder mostrar el recurso en ninguno de los formatos indicados por el cliente mediante el campo Accept del encabezado, devolverá el código de estado **HTTP 406**.

Petición

=====

GET /facturas/123

Accept: application/epub+zip , application/pdf, application/json

Respuesta

=====

Status Code 200

Content-Type: application/pdf

- Permiten que empresas ubicadas en diferentes lugares geográficos combinen fácilmente servicios y software para proporcionar servicios integrados.
- Aportan interoperabilidad entre las aplicaciones de software sin tener en cuenta sus propiedades o las plataformas sobre las que se instalen.
- Son fáciles de entender (su contenido y funcionamiento) debido a que fomentan los protocolos y formatos para el intercambio de datos.
- Pueden aprovechar los sistemas de seguridad firewall (se apoyan en HTTP) sin necesidad de cambiar las reglas de filtrado.

- Facilitan la integración con afiliados de negocio, al poder compartir servicios internos con un alto grado de integración.
- Disminuyen el tiempo de desarrollo de las aplicaciones.
- No están ligados a ningún Sistema Operativo o Lenguaje de Programación.
- No necesitan usar browsers (navegadores) ni el lenguaje de especificación HTML.
- Permite el cambio de la lógica de presentación de manera sencilla, debido a que su arquitectura ofrece la alternativa de separar por completo la lógica de presentación, lógica de negocio y el almacenamiento de los datos.

Muchas gracias

Preguntas

