Programación de Servicios y Procesos

Servicios

IES Nervión
Miguel A. Casado Alías

Sistema distribuido

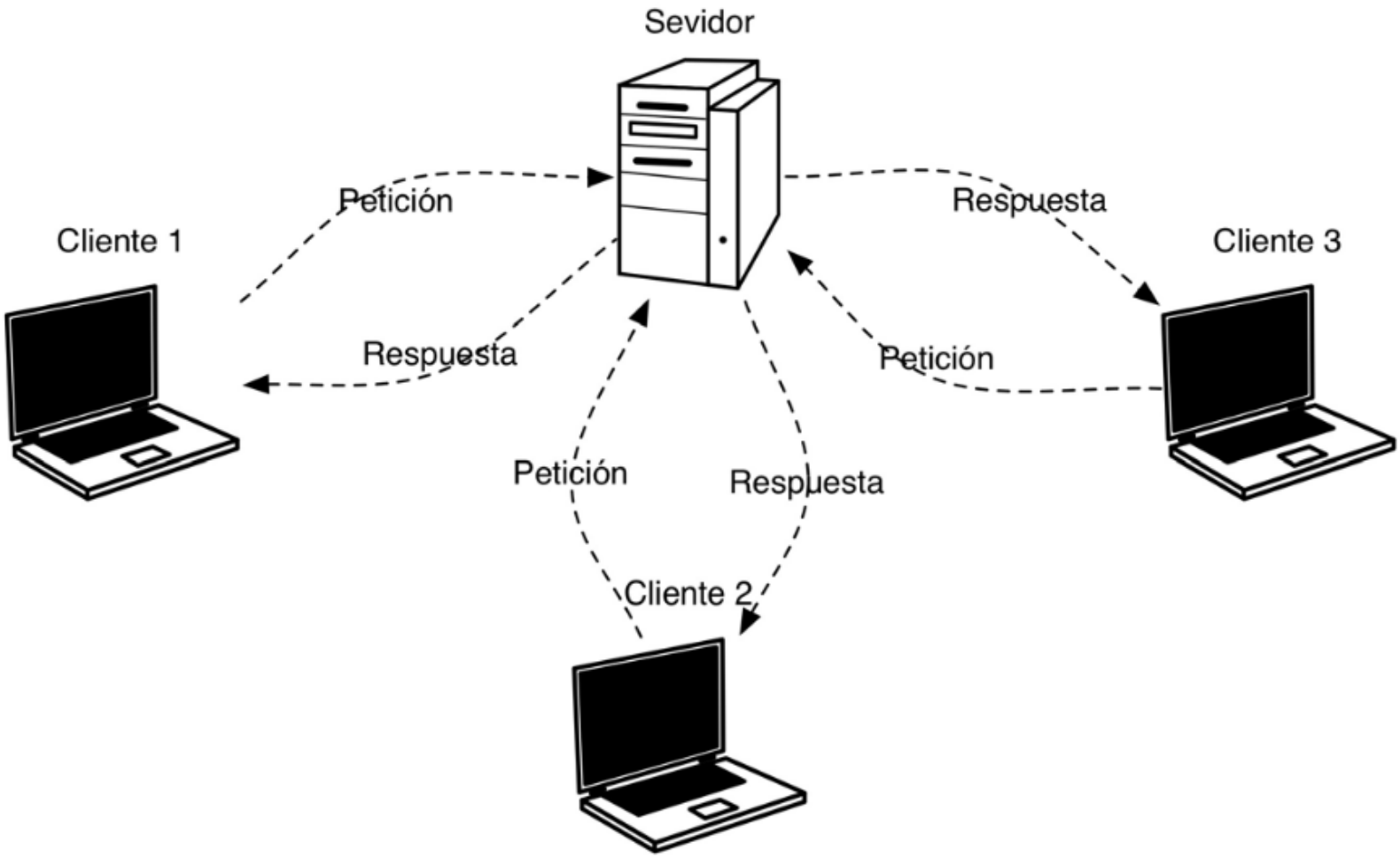
- Formado por diversos elementos computacionales, independientes, pero que se comunican entre sí. Ejemplos:
  - Procesos independientes que se ejecutan dentro de una misma máquina y colaboran entre sí intercambiando mensajes
  - Ordenadores separados físicamente, conectados entre sí a través de una red de comunicaciones

- Los usuarios ven el sistema como una sola unidad. Los detalles de implementación (múltiples procesos, múltiples computadores…) no se perciben externamente.

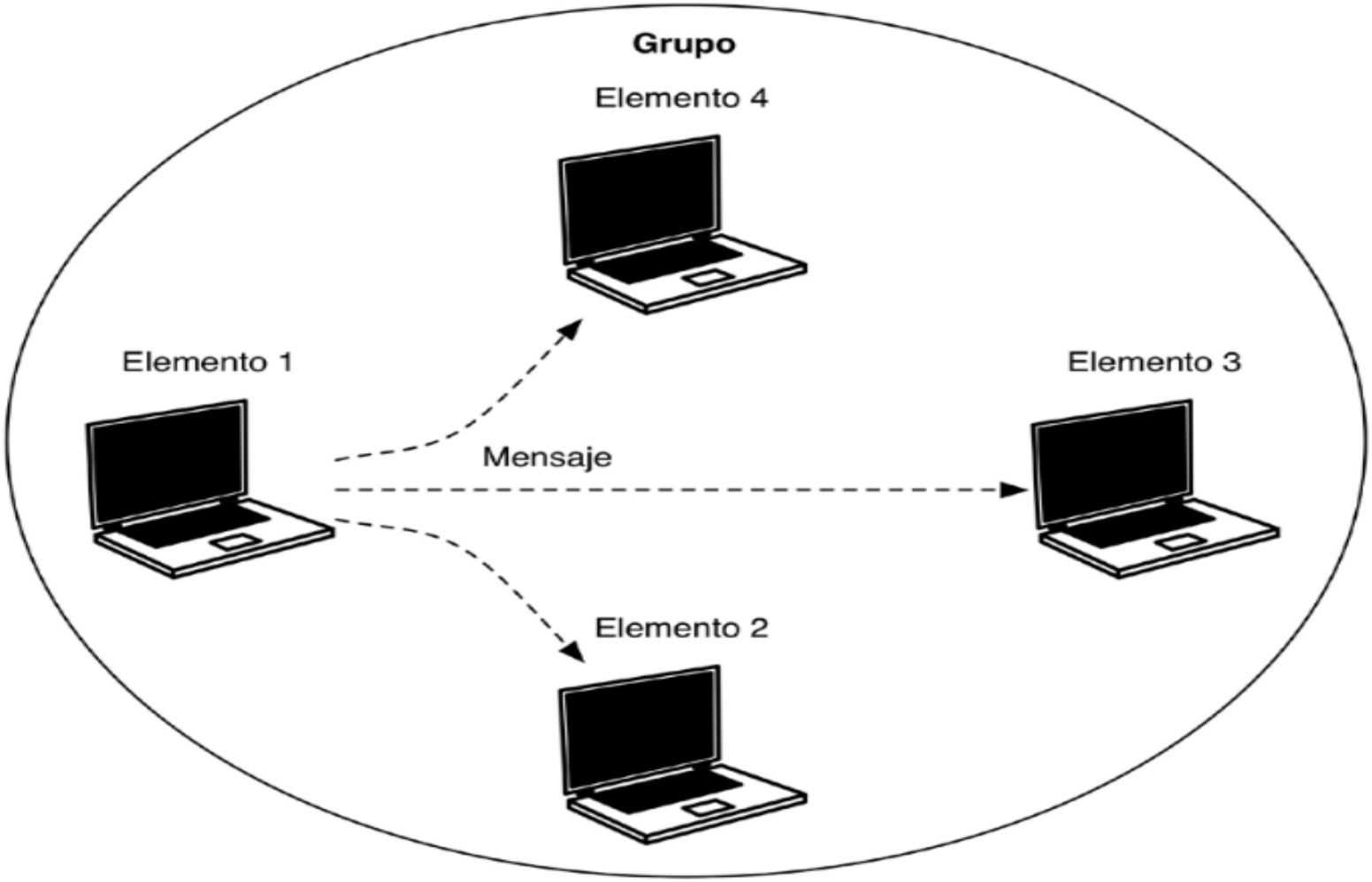
Modelo de comunicaciones

- Arquitectura general que especifica cómo se comunican entre sí los diferentes elementos de una aplicación distribuida
  - Cliente – Servidor
    - El más utilizado
    - Un proceso central (servidor) ofrece una serie de **servicios** a uno o más procesos clientes
    - Cuando un cliente requiere un servicio, hace una petición al servidor, el cual responde a la solicitud
  - Comunicación en grupo
    - No existen roles diferenciados ni jerarquías
    - Comunicación mediante multidifusión (multicast)

Cliente - Servidor



Comunicación en grupo



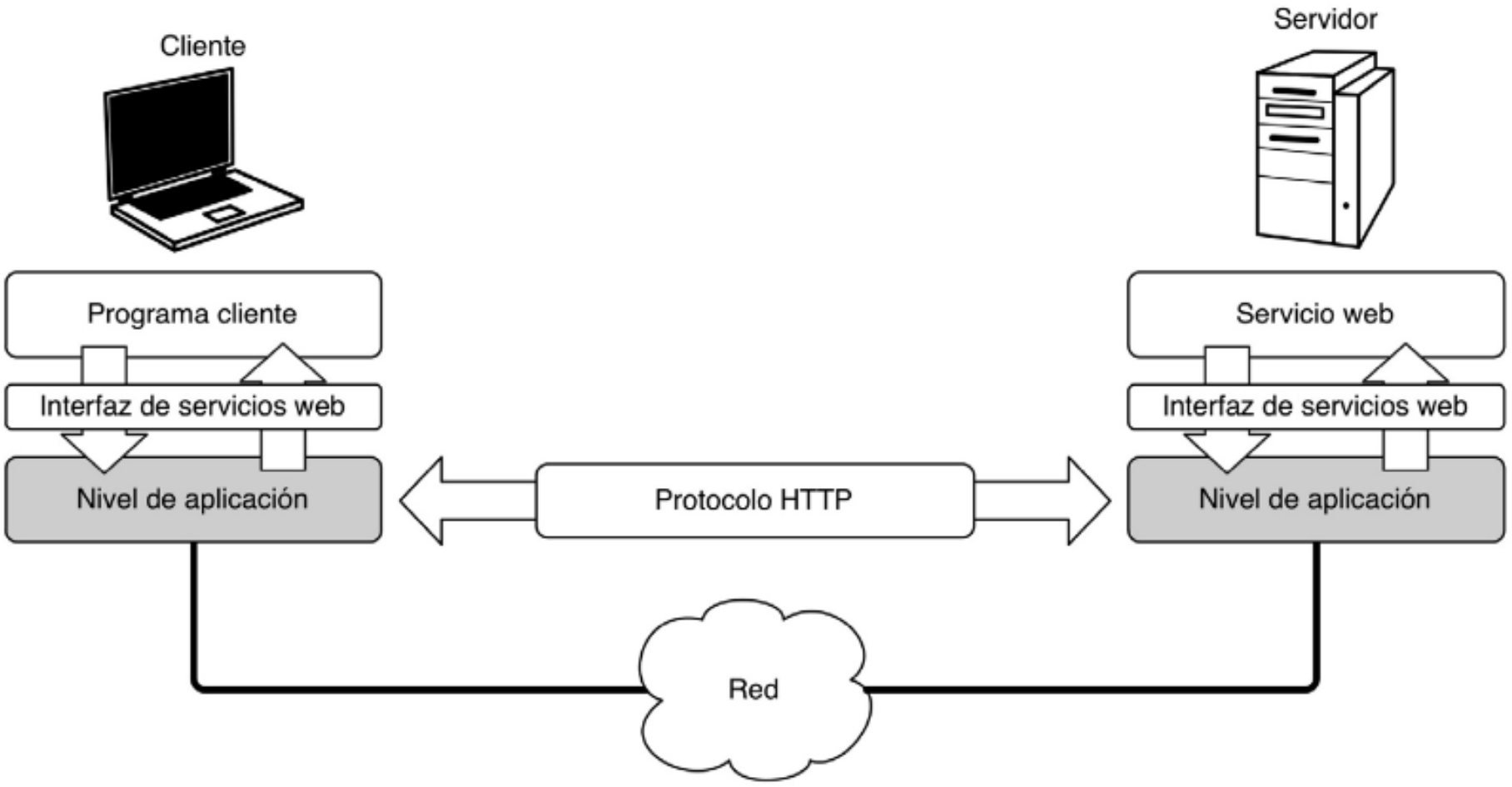
Algunos protocolos estándar de nivel de aplicación (Cliente - Servidor)

- Telnet: Conexión a máquinas remotas. No seguro.
- SSH: Secure Shell. Conexión remota de forma segura
- FTP: Transferencia de ficheros
- HTTP: Transferencia de documentos de hipertexto
- POP3: Descarga de correo electrónico
- SMTP: Envío de correo electrónico
- DNS: Localiza IP a partir de un nombre de dominio
- DHCP: Configuración de máquinas en una red local
- NTP: Network Time Protocol

¿Qué es un servicio? ¿Y un servicio web?

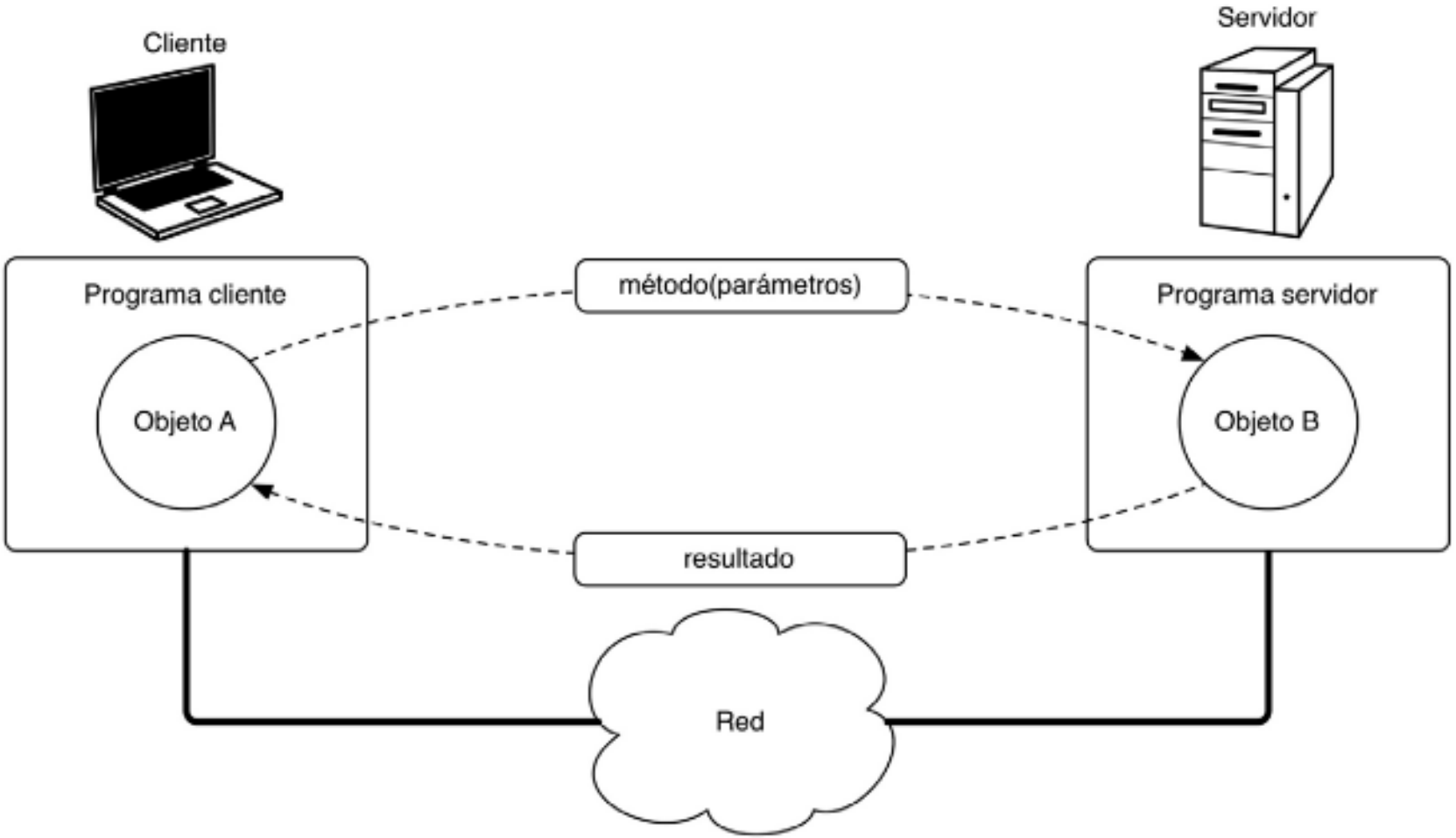
- Es un conjunto de funcionalidades software que son ofrecidas a los clientes para que hagan uso de ellas
- Servicio web:
  - Accesible en remoto a través de la web (HTTP)
  - Interfaz bien definida, ocultando implementación real
  - Interoperable: El servidor y el cliente pueden estar implementados en tecnologías distintas y aún así poder interactuar

Servicio Web



Servicios RPC

- Remote Procedure Call
- Orientado a acciones (p.ej: obtenerCoches, borrarAlumno)



Servicios REST

- **Re**presentational **S**tate **T**ransfer
- Es un conjunto de restricciones a respetar cuando diseñamos nuestros servicios web:
  - Es orientado a recursos, no a acciones (RPC)
    - Se publican nombres (p.ej: “Cliente con id 32”) en lugar de verbos (p.ej: “Obtener cliente con id 32”)
  - Cada recurso posee un identificador único universal (UUID)
  - La forma en que un recurso se representa internamente no es accesible desde el exterior
  - Cada recurso admite un conjunto de operaciones (interfaz). Las operaciones soportadas por cada recurso se deben escoger de un conjunto cerrado, no podemos inventar nuevas operaciones. P.ej: “Leer”, “Actualizar”, “Crear”; NO podríamos inventar la operación “Leer clientes de más de 30”

Servicios REST (II)

- Las operaciones se realizan mediante la transferencia del estado del recurso entre cliente y servidor
- Las operaciones no tienen estado (stateless). Cada petición se trata como una transacción independiente, que no tiene relación con cualquier petición anterior
- El estado de un recurso puede ser representado mediante diversos formatos (XML, JSON, PDF...)

REST en la práctica: HTTP

- HTTP es un protocolo que cumple los principios REST:
  - Orientado a recursos: cada página web es un recurso
  - Cada recurso tiene un UUID: su URI
  - Operamos sobre los recursos mediante verbos HTTP (GET, POST…)
  - Las operaciones son transferencias de los estados de los recursos (p.ej: para leer una página web, se envía una petición GET sobre una URI, a lo que el servidor responde enviando el documento HTML que es la representación del estado del recurso identificado por dicha URI)
  - Cada página puede soportar varias representaciones (aunque no es lo habitual). HTTP dispone de cabeceras (Accept, Content-Type…) para especificar el formato de representación que entienden servidor y cliente
- Para hacer servicios web, lo ideal es respetar HTTP => REST

Mensaje de petición HTTP

- Primera línea: Verbo + URI + Versión
- Después cabeceras (opcionales). Una por línea.
  - Nombre_cabecera : Valor
- Tras las cabeceras, el cuerpo (separado de cabeceras por línea en blanco). Es opcional
  - Contiene un documento
  - Cualquier formato, aunque lo habitual es texto plano (HTML, XML, JSON…)
  - El formato se define con la cabecera **Content-Type**

Mensaje de respuesta HTTP

- Primera línea: Versión + Código respuesta + Explicación textual de la respuesta
- Después cabeceras (opcionales). Una por línea.
  - Nombre_cabecera : Valor
- Tras las cabeceras, el cuerpo (separado de cabeceras por línea en blanco). Es opcional
  - Contiene un documento
  - Cualquier formato, aunque lo habitual es texto plano (HTML, XML, JSON…)
  - El formato se define con la cabecera **Content-Type**

Ejemplo de petición HTTP

```
POST /server/payment HTTP/1.1
Host: www.myserver.com
Content-Type: application/x-www-form-urlencoded
Accept: application/json
Accept-Encoding:gzip,deflate,sdch
Accept-Language:en-US,en;q=0.8
Cache-Control:max-age=0
Connection:keep-alive

orderId=34fry423&payment-method=visa&card-number=2345123423487648&sn=345
```

Ejemplo de respuesta HTTP

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=utf-8
Location: https://www.myserver.com/services/payment/3432
Cache-Control:max-age=21600
Connection:close
Date:Mon, 23 Jul 2012 14:20:19 GMT
ETag:"2cc8-3e3073913b100"
Expires:Mon, 23 Jul 2012 20:20:19 GMT

{
"id":"https://www.myserver.com/services/payment/3432",
"status": "pending"
}
```

Verbos HTTP

- Seguros: Verbos que no provocan efectos secundarios (modificaciones) en el estado del servidor.
- Idempotentes: Verbos que aunque se ejecuten repetidamente, con los mismos parámetros, tienen el mismo efecto que si solo se hubieran ejecutado una vez
- Los verbos HTTP más habituales:

Método	Seguro	Idempotente	Semántica
GET	Sí	Sí	Leer el estado del recurso
HEAD	Sí	Sí	Leer, pero sólo las cabeceras
PUT	No	Sí	Actualizar o crear
DELETE	No	Sí	Eliminar un recurso
POST	No	No	Cualquier acción genérica no idempotente
OPTIONS	Sí	Sí	Averiguar las opciones de comunicación disponibles de un recurso

Tipos MIME en HTTP

- Formatos a usar en la transferencia del estado entre servidor y cliente
- http://www.iana.org/assignments/media-types/media-types.xhtml
- Se expresan como <tipo>/<subtipo>. Por ejemplo:
  - application/json, video/mp4, text/html
- En la petición, con la cabecera **Accept**, el cliente le indica al servidor los tipos MIME que soporta
- El servidor responde con uno de los tipos MIME que el cliente solicitó o con el **código 406** si no soporta ninguno
- La cabecera **Content-Type** determina el tipo MIME del cuerpo del mensaje
- El parámetro **q** indica orden de preferencia. P.ej:
  - Accept: text/*;q=0.3, text/html;q=0.7

Códigos de respuesta HTTP

- Códigos **1xx**: Respuestas **informativas**. Indica que la petición ha sido recibida y se está procesando
- Códigos **2xx**: Respuestas **correctas**. Indica que la petición ha sido procesada correctamente
- Códigos **3xx**: Respuestas de **redirección**. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- Códigos **4xx**: **Errores causados por el cliente**. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.
- Códigos **5xx**: **Errores causados por el servidor**. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor
- https://es.wikipedia.org/wiki/Anexo:Códigos_de_estado_HTTP

Servicios REST de tipo CRUD

- **C**reate, **R**ead, **U**pdate, **D**elete
- Es el tipo de servicio REST más sencillo de diseñar
- Hace referencia a operaciones de mantenimiento de datos, normalmente sobre tablas de una BD
- Tendremos dos tipos de recursos:
  - Colecciones: listas o contenedores de entidades. Se suelen corresponder con una tabla de la BD
  - Entidades: ocurrencias o instancias concretas dentro de una colección. Se suelen corresponder con un registro de una tabla.

Colecciones

- Las identificaremos con una URI derivada del nombre de la entidad que contienen. Por ejemplo:

http://www.server.com/rest/libro

- Dicha colección representaría todos los libros del sistema
- Se suele usar la siguiente correspondencia entre métodos HTTP y operaciones:

Método HTTP	Operación
GET	Leer todas las entidades dentro de la colección
PUT	Actualización mútiple y/o masiva
DELETE	Borrar la colección y todas sus entidades
POST	Crear una nueva entidad dentro de la colección

Entidades

- Las identificaremos concatenando a la URI de la colección correspondiente un identificador de entidad. Por ejemplo:

  http://www.server.com/rest/libro/AF74gt0

- Representaría al libro con identificador AF74gt0
- Se suele usar la siguiente correspondencia entre métodos HTTP y operaciones:

Método HTTP	Operación
GET	Leer los datos de una entidad en concreto
PUT	Actualizar una entidad existente o crearla si no existe
DELETE	Borrar una entidad en concreto
POST	Añadir información a una entidad ya existente

Entidades con recursos hijos (detalles)

- A veces el volumen de información de una entidad es muy elevado y puede ser buena idea dividirla en recursos hijos
- Por ejemplo, si de un libro queremos ofrecer además de autor, título, precio, ISBN y número de páginas, la información de todos sus capítulos, podríamos hacer una colección de capítulos dentro de libro, con URIs como:
  - /rest/libro/Ae3Fsr7/capitulo información de todos los capítulos
  - /rest/libro/Ae3Fsr7/capitulo/3 nos daría info del capítulo 3
- Al hacer GET sobre /rest/libro/Ae3Fsr7 se ofrecerá la información de autor, título, precio, ISBN, número de páginas y las URIs de todos los capítulos del libro
- También podemos dar información parcial de una entidad sin diseñar recursos hijos => usando query strings:
  - GET /rest/libro/Ae3Fsr7?capitulo=3 (es menos recomendable)

Lectura de recursos

- Usaremos el verbo GET tanto para leer colecciones como entidades
- Para obtener todos los miembros de una colección, hacemos GET sobre la URI de la colección. ¿Qué devolveremos? Dos opciones:
  - Lista con enlaces (URI) a todas las entidades
  - Lista de entidades con todos sus datos

```
1  GET /rest/libro HTTP/1.1
2  Host: www.server.com
3  Accept: application/json
4
```

Respuesta a GET con solo enlaces

```
1  HTTP/1.1 200 Ok
2  Content-Type: application/json;charset=utf-8
3
4  ["http://www.server.com/rest/libro/45",
5   "http://www.server.com/rest/libro/465",
6   "http://www.server.com/rest/libro/4342"]
```

Respuesta a GET con toda la información

```
HTTP/1.1 200 Ok
Content-Type: application/json;charset=utf-8

[ {
    "id": "http://www.server.com/rest/libro/45",
    "author": "Rober Jones",
    "title": "Living in Spain",
    "genre": "biographic",
    "price": { "currency": "$", "amount": 33.2}
  },
  {
    "id": "http://www.server.com/rest/libro/465",
    "author": "Enrique Gómez",
    "title": "Desventuras de un informático en Paris",
    "genre": "scifi",
    "price": { "currency": "€", "amount": 10}
  },
  {
    "id": "http://www.server.com/rest/libro/4342",
    "author": "Jane Doe",
    "title": "Anonymous",
    "genre": "scifi",
    "price": { "currency": "$", "amount": 4}
  } ]
```

Filtrar una colección

- Podemos hacer uso de query strings cuando no queramos obtener todas las entidades de una colección, sino solo las que cumplen algunos requisitos.

```
1  GET /rest/libro?precio_max=20eur&genero=scifi HTTP/1.1
2  Host: www.server.com
3  Accept: application/json
4
```

Actualizar

- Tenemos dos opciones:
  - PUT: actualiza el contenido de un recurso, y si éste no existe crea un nuevo recurso en la URI especificada
  - POST: se puede usar para cualquier operación no segura ni idempotente, normalmente para añadir un trozo de información a un recurso o para crear un nuevo recurso
- Para actualizar una entidad, lo más sencillo es hacer PUT sobre la URI de la misma e incluir en el cuerpo de la petición HTTP los nuevos datos
- PUT realiza una actualización de manera que se reemplazan por completo los datos del servidor con los que enviamos en la petición
- Se puede usar PUT sobre una colección. Los datos de la colección pasarían a ser los que mandemos en el cuerpo

Actualizar entidad con PUT

Petición:

```
PUT /rest/libro/465 HTTP/1.1
Host: www.server.com
Accept: application/json
Content-Type: application/json

{
  "author": "Enrique Gómez Salas",
  "title": "Desventuras de un informático en Paris",
  "genre": "scifi",
  "price": { "currency": "€", "amount": 50}
}
```

Respuesta:

```
HTTP/1.1 204 No Content

```

Actualización parcial

- No podemos usar PUT si solo queremos enviar en la petición los campos a cambiar de uno o varios recursos
- Según HTTP, al usar PUT, el contenido del cuerpo de la petición pasa a ser el nuevo estado del recurso
- En estos casos mejor usar POST, ya que se puede interpretar como añadir contenido a un recurso
- POST se podría interpretar de otras formas, así que mejor combinarlo con un tipo MIME “diff” que indique que lo que se envía es un fragmento de una entidad que se añade al recurso del servidor
- CUIDADO => POST no es idempotente. Si se usa repetidas veces podemos volver a aplicar cambios

Ejemplo petición actualización parcial

```
1  POST /rest/libro/465 HTTP/1.1
2  Host: www.server.com
3  Accept: application/json
4  Content-Type: application/book-diff+json
5
6  [{
7      "change-type":"replace",
8      "location":"price/amount",
9      "value":100
10  },
11  {
12    "change-type":"append",
13    "location":"author",
14    "value":"Juan Pérez"
15  }]
```

Problema con NO idempotencia de POST

- Si el servidor no está bien diseñado:

```
1  HTTP/1.1 200 Ok
2  Content-Type: application/json;charset=utf-8
3
4  {
5    "id": "http://www.server.com/rest/libro/465",
6    "author": [ "Enrique Gómez Salas", "Juan Pérez", "Juan Pérez" ],
7    "title": "Desventuras de un informático en Paris",
8    "genre": "scifi",
9    "price": { "currency":"€", "amount":100}
10 }
```

- Debería responder:

```
1  HTTP/1.1 409 Conflict
2
```

Concurrencia optimista

- Puede pasar que dos clientes intenten hacer modificaciones simultáneamente al mismo recurso
- Para evitar inconsistencias se puede hacer uso de las cabeceras If-Match y Etag
- El servidor manda en la cabecera Etag un hash del estado del recurso cuando se consulta. El hash solo cambiará si el recurso cambia
- La cabecera If-Match nos permite poner como precondición que la petición de actualización solo se lleve a cabo si el hash (ETag) del recurso no ha cambiado con respecto al hash que conocía el cliente

https://looselyconnected.wordpress.com/2010/03/25/the-http-etag-header-and-optimistic-locking-in-rest/

Actualización parcial con PATCH

- PATCH es un verbo HTTP más novedoso
- Es NO seguro y NO idempotente, al igual que POST
- Está expresamente pensado para actualizaciones parciales, a diferencia de POST que se puede emplear para múltiples propósitos
- Se usa de la misma manera que hemos visto antes en la actualización parcial con POST, simplemente cambiamos POST por PATCH, que es más específico
- La desventaja de PATCH es que al ser más nuevo, podemos tener problemas de interoperabilidad: dispositivos que no lo implementen, firewalls que lo bloqueen...

Borrar

- Se usa el método DELETE
- Al borrar una colección, se deben borrar todas sus entidades
- Al borrar una entidad, se deben borrar sus entidades hijas
- Con query strings hacemos borrado selectivo

Peticiones:

```
DELETE /rest/libro/465 HTTP/1.1
Host: www.server.com

```

```
DELETE /rest/libro?genero=scifi HTTP/1.1
Host: www.server.com

```

Respuesta:

```
HTTP/1.1 204 No Content

```

Crear entidades

- Podemos crear una entidad haciendo PUT sobre una URI que no esté asociada a ninguna, y mandando los datos del nuevo recurso.
- También podríamos crearla haciendo POST sobre una colección
- POST tiene la ventaja de que la lógica de creación de URIS no recae en el cliente. Además, implica que la entidad se asocia a la colección (en PUT no tendría porqué ser así). Problema de POST: NO es idempotente.
- El servidor responde con código 201 en caso de éxito
- La cabecera Location lleva la URI del recurso creado

Ejemplo petición crear con PUT

```
1  PUT /rest/libro/465 HTTP/1.1
2  Host: www.server.com
3  Accept: application/json
4  Content-Type: application/json
5
6  {
7    "author": "Enrique Gómez Salas",
8    "title": "Desventuras de un informático en Paris",
9    "genre": "scifi",
10   "price": { "currency": "€", "amount": 50}
11 }
```

Ejemplo respuesta petición crear con PUT

```
1  HTTP/1.1 201 Created
2  Location: http://www.server.com/rest/libro/465
3  Content-Type: application/json;charset=utf-8
4
5  {
6    "id": "http://www.server.com/rest/libro/465",
7    "author": "Enrique Gómez Salas",
8    "title": "Desventuras de un informático en Paris",
9    "genre": "scifi",
10   "price": { "currency": "€", "amount": 50}
11 }
```

Ejemplo petición crear con POST

```
1  POST /rest/libro HTTP/1.1
2  Host: www.server.com
3  Accept: application/json
4  Content-Type: application/json
5
6  {
7    "author": "Enrique Gómez Salas",
8    "title": "Desventuras de un informático en Paris",
9    "genre": "scifi",
10   "price": { "currency": "€", "amount": 50}
11 }
```

Ejemplo respuesta petición crear con POST

```
1  HTTP/1.1 201 Created
2  Location: http://www.server.com/rest/libro/3d7ef
3  Content-Type: application/json;charset=utf-8
4
5  {
6    "id": "http://www.server.com/rest/libro/3d7ef",
7    "author": "Enrique Gómez Salas",
8    "title": "Desventuras de un informático en Paris",
9    "genre": "scifi",
10   "price": { "currency": "€", "amount": 50}
11 }
```

Procesos de negocio

HATEOAS – Hypermedia APIs