Tema 1. Desarrollo de APIs REST con Spring Boot.

1. PROTOCOLO HTTP

HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web.

HTTP es un protocolo **sin estado**, es decir, no guarda ninguna información sobre conexiones anteriores. Para guardar información se usan las **cookies**, que guardan la información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de sesión,

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

Estructura de un mensaje

- **Línea inicial** (termina con un retorno de carro y salto de línea) con:
  - *Peticiones*: acción requerida al servidor, URL del recurso y versión HTTP soportada.
  - *Respuestas*: versión HTTP usada, código de respuesta y la URL del recurso.

- Las **cabeceras**, que terminan con una línea en blanco. Son metadatos que aportan gran flexibilidad al protocolo.

- **Cuerpo del mensaje** (Opcional): Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL.
  Contiene los datos que se intercambian cliente y servidor.

Métodos (Verbos)

- GET, POST, PUT, DELETE

Códigos de Respuesta

- **1xx** -> Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.

- **2xx** -> Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- **3xx** -> Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- **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.
- **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.

Cabeceras

Son los metadatos que se envían en las peticiones o respuesta HTTP para proporcionar información esencial sobre la transacción en curso.
Cada cabecera es especificada por un nombre de cabecera seguido por dos puntos, un espacio en blanco y el valor de dicha cabecera seguida por un retorno de carro seguido por un salto de línea.

Dependiendo del tipo de mensaje en el que puede ir una cabecera las podemos clasificar en cabeceras de petición, cabeceras de respuesta y cabeceras que pueden ir tanto en una petición como en una respuesta.

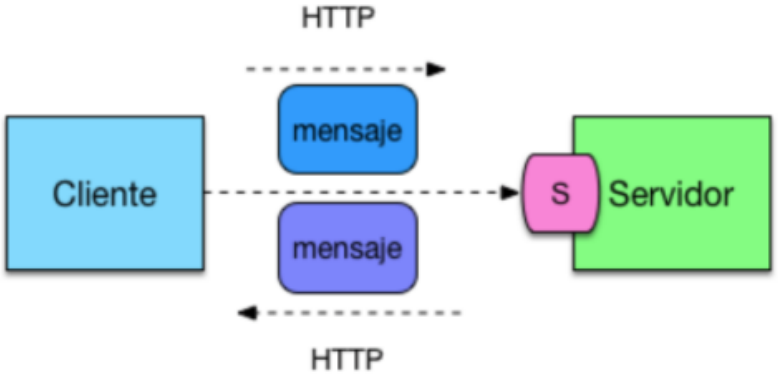
Podemos clasificar las cabeceras según su función: Las que indican las capacidades aceptadas, las que definen el contenido, las que hacen referencia a URLs, cabeceras de control de cookies...

2. REST

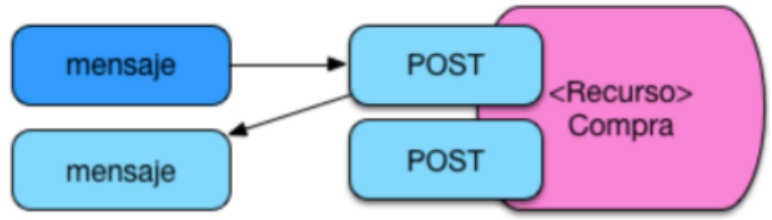
Es un estilo de arquitectura a la hora de realizar una comunicación entre cliente y servidor.

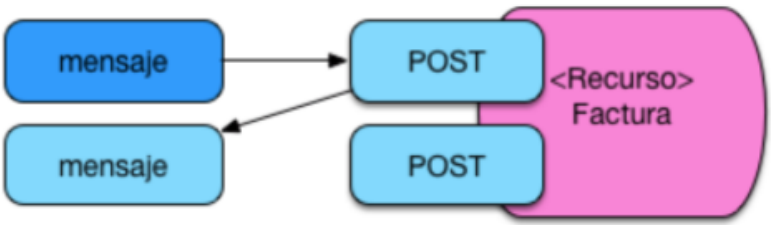
NIVELES

- *Nivel 0:* Se envían mensajes sin ningún tipo de organización.

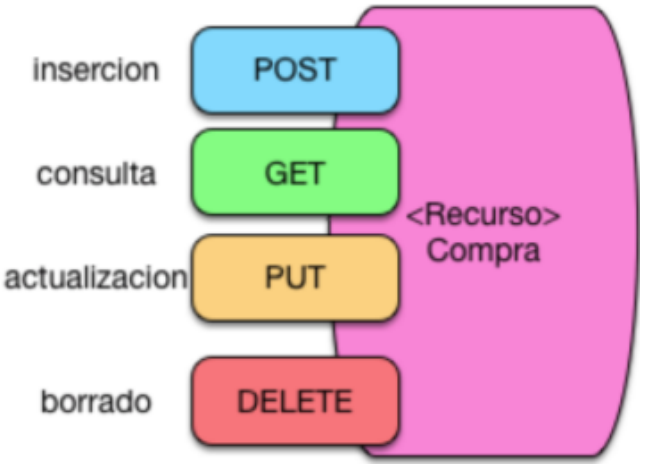


- *Nivel 1:* Declaramos recursos en vez de métodos. Un recurso es un concepto importante del negocio, también llamado objeto de negocio (facturas, cursos, compras, etc.).





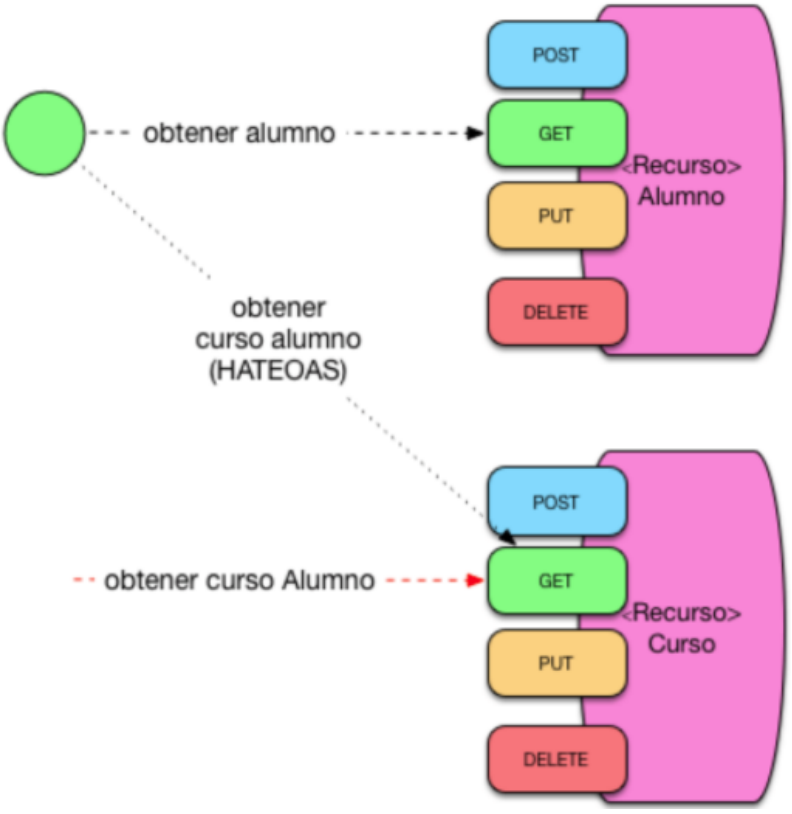
- *Nivel 2:* Verbos HTTP. Operaciones categorizadas más estrictamente según el tipo de operación. Nivel más usado actualmente por la mayoría de las API de red.



Dependiendo del tipo de operación se utilizará un método diferente

- **GET -> 200**
  - **/** -> Listado de pedidos
  - **/id** -> Información de un pedido
- **POST** -> 201
- **PUT + /id** -> 200
- **DELETE + /id** -> 204

- *Nivel 3:* HATEOAS. Permite que un usuario reciba en la respuesta un enlace al recurso en vez del código del recurso, facilitando el uso de la aplicación y encadenando una petición con otra, si fuera necesario. Sin embargo, su implementación es más complicada.



Podríamos tener una estructura JSON como la que se muestra:

```
{
    nombre:"pedro",
    apellidos:"gomez",
    cursos:"http://miurl/cursos"
}
```

3. JSON

NOTAS

Usando la anotación **@lob** podemos convertir un String en un objeto grande, de forma que se convierta en el tamaño máximo que pueda albergar la base de datos. Adicionalmente, podemos usar la anotación **@Column(length=x)**.

Si en el método de la petición POST declaramos que va a devolver la clase entidad directamente, siempre se nos devolverá un código de respuesta 200. Para que se nos devuelva un código 201, hay que utilizar la clase **ResponseEntity<T>**.

@RequestBody indica que en el cuerpo de la petición se debe incluir el objeto que se va a crear.