**API REST**

Un servicio REST ofrece operaciones CRUD (creación, lectura, actualización y borrado) sobre recursos (items de información) del servidor web, el cual se aprovecha de todos los aspectos del protocolo http y dicha información se intercambia en formato JSON o XML.

Para implementar una API REST con Java se puede usar:

* JAX-RS
  + Estándar JEE.
  + Java API for RESTful Web Services.
* Spring MVC
  + Framework Spring (no estándar).
  + Mismo sistema usado para aplicaciones web.

Niveles de cumplimiento de los principios REST

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

El enfoque más habitual en los servicios REST es el nivel 2.

* Los recursos se identifican en la URL. Una parte de la URL es fija y la otra apunta al recurso concreto.
* Las operaciones que se quieren realizar con ese recurso son los métodos del protocolo HTTP.
* Se usan los códigos de estado http para notificar errores.

**API REST con SpringMVC**

Se crea un controlador con la anotación @RestController. Esta clase será la encargada de gestionar las peticiones que se hagan a nuestra API e indica que los datos devueltos por cada método se escribirán directamente en el cuerpo de la respuesta.

Se implementa un método en la clase por cada URL de la API REST:

* Se anota con @RequestMapping para indicar la URL y el verbo HTTP correspondiente (GET, POST, etc.) se encarga de representar los endpoints de nuestra API.
* El método devuelve el objeto que quiere enviar al cliente.

Cuando todas las URLs de un controlador empiezan de forma similar, se puede poner la anotación @RequestMapping a nivel de clase con la parte común. Por defecto el verbo es GET.

En caso de querer insertar información en la URL nos encontramos con:

**URL con parámetros**

Es habitual que se incluya información en la URL para que esté disponible en el servidor cuando el usuario pulsa el enlace.

* Los parámetros se incluyen al final de la URL separados con ? (query).
* Los parámetros se separan entre sí con &
* Cada parámetro se codifica como nombre=valor

Para acceder a la información se usa @RequestParam

**Como parte de la propia URL**

La información también se puede incluir como parte de la propia URL, en vez de cómo parámetros.

El “name” del recurso se codifica en la ruta y se accede a él usando un @PathVariable, en caso de querer añadir un nuevo recurso, indicamos que el método atiende peticiones POST

**Nuevas anotaciones**

Actualmente, Spring permite cinco tipos de anotaciones para manejar los diferentes tipos de métodos de solicitud HTTP. Estas anotaciones son:

* @GetMapping.
* @PostMapping.
* @PutMapping.
* @DeleteMapping.
* @PatchMapping.

**Clase ResponseEntity**

En ciertos momentos se necesita enviar respuestas HTTP desde nuestro backend hacia el cliente. Una de las maneras en la que Spring trabaja con ello es usar ResponseEntity para manejar el cuerpo, cabeceras y el estado de las respuestas.

Si el recurso existe se devuelve, y si no, se devuelve 404 NOT FOUND. Por eso el método devuelve un ResponseEntity. Finalmente, ResponseEntity provee dos clases anidadas de tipo interface: BodyBuilder y HeadersBuilde

**Solucionar problema CORS**

Por razones de seguridad, los navegadores prohíben las llamadas AJAX a recursos fuera del origen actual. El intercambio de recursos de origen cruzado (CORS) es una especificación W3C, implementada por la mayoría de navegadores, que permite especificar qué tipo de solicitudes están autorizadas.

Para habilitar las solicitudes de origen cruzado es necesario tener alguna configuración CORS explícitamente declarada. El HandlerMapping de Spring MVC proporciona soporte para el CORS. Cada HandlerMapping puede ser configurado individualmente. Después de mapear con éxito una petición a un handler, se comprueba la configuración del CORS, la cual es interceptada y validada.

Se puede combinar la configuración CORS global a nivel de HandlerMapping con una configuración más fina a nivel de clase o de método, por ejemplo, con la anotación @CrossOrigin.