**DISEÑO DE APIs REST**

**PROTOCOLO HTTP**

HTTP es un protocolo sin estado, no guarda ninguna información sobre conexiones anteriores.

* Petición: El cliente envía una petición al servidor. Está compuesta por:
* URI (obligatoria): dominio del servidor + puerto + ruta del recurso + parámetros del recurso (opcional).
* Header (cabecera): Establece la acción a realizar con los verbos HTTP (obligatorios) y puede contener parámetros (opcional)
* Body (cuerpo): Opcional. Con los métodos GET y DELETE no se manda cuerpo. Solo se mandan con POST, PUT y PATH.
* Respuesta: Envío de la respuesta del servidor al cliente
* Header (encabezado): Devuelve el estado de la petición (obligatorio)
  + 1xx: Respuesta informativa
  + 2xx: Peticiones correctas
  + 3xx: Redirecciones
  + 4xx: Errores del cliente
  + 5xx: Errores internos (servidor)
  + Parámetros
* Body (cuerpo): Opcional.

**VERBOS REST**

Indican la acción a realizar sobre el recurso deseado.

|  |  |  |  |
| --- | --- | --- | --- |
| **Verbo** | **Seguro** | **idempotente** | **Respuesta** |
| **GET** | Sí | Sí | 200 (OK) o tipo de error  Devolver el recurso o un conjunto |
| **POST** | No | No | 201 (OK) o tipo de error  Recomendable devolver el nuevo recurso o una referencia al mismo recurso |
| **PUT** | No | Sí | 200 (OK) o tipo de error  Recomendable devolver el recurso |
| **PATCH** | No | Sí | 200 (OK) o tipo de error  Recomendable devolver el recurso |
| **DELETE** | No | Sí | 204 (OK) sin contenido o tipo de error |
| **HEAD** | Sí | Sí |  |
| **OPTIONS** | Sí | Sí |  |

* **GET**: Conculta sobre un recurso (colección o entidad).
* **POST**: Alta o creación de nuevos recursos o entidades sin conocer la URI (se invocado sobre un recurso padre)
* **PUT**: Crea recursos en la URI especificada y/o modifica un recurso ya creado, de forma idempotente.
* **DELETE**: Elimina recursos

**IDEMPOTENCIA**

Es la invocación repetida sobre el mismo recurso con los mismos parámetros tiene el mismo efecto en el servidor independientemente del número de invocaciones. Ello no conlleva que la respuesta sea la misma.

La idempotencia es útil cuando se quiere garantizar la entrega de una petición, en caso de que falle la comunicación. Si se repite la petición, como no se altera el estado del servidor, se garantiza que la acción ha tenido efecto una única vez, independientemente del número de veces que se haya repetido. Si se usara un verbo no idempotente, la repetición de la solicitud podría haber alterado el estado del servidor múltiples veces.

**RECUROS VIRTUALES**

Se envía una petición con un verbo no idempotente (POST). El servidor devuelve una URL virtual, que aún no ha creado pero que está disponible o es válida. A continuación, se envía la petición con un verbo idempotente (PUT) con la URL virtual. El servidor entonces crea el recurso y su URL, devolviendo la confirmación al cliente.

**ENDPOINT**

Se define por un método o verbo HTTP y una URI indicando el recurso sobre el que se actúa.

* Seguro: Cuando no cambia el estado ni provoca efectos secundarios, es decir, no se cambia la base de datos.
* Idempotente: Cuando la operación lanzada de forma repetida devuelve el mismo resultado.

**HATEOAS**

Hypermedia As The Engine Of Application State. Los recursos están interconectados entre sí. Ante una petición, se devuelve enlaces a otro tipo de recursos con las que está relacionado el recurso de la petición, de forma que para el cliente o usuario, la aplicación es autodescubrible, ya que puede ir explorándola sin necesidad de realizar peticiones específicas para ello.

El principio HATEOAS establece que una API, a partir de un recurso raiz, pueda ser recorrida sin necesidad de ninguna documentación.

Ventajas:

* Aumenta la interoperabilidad, facilitando la implementación de la API con clientes de diferentes naturaleza.
* En teoría, se puede ahorrar documentación, al ser autodescubrible no es necesario extender tanto el “manual”.
* Ayuda a reducir la cantidad de datos que devuelve el servidor, ya que una petición que requiere gran volumen de datos, pueden devolverse los principales y enlaces al resto de datos.

**NIVELES DE MADUREZ REST**

Establecen unos criterios o baremos que evaluar en qué medida una API cumple con los principios REST y sus reglas de “buen diseño”, de forma que el nivel 0 sería el cumplimiento más básico (poca madurez de laAPI) y el nivel 3 sería un cumplimiento ideal (API madura).

* Nivel 0: HTTP como transporte, es decir, se utiliza el protocolo HTTP como canal para enviar peticiones y devolver respuestas.
* Nivel 1: El diseño se orienta a recursos y no a acciones o métodos.
* Nivel 2: Uso correcto de verbos HTTP, eliminando otras sintaxis de acción incluidas en la URL.
* Nivel 3: Uso de controles hipermedia, de acuerdo al principio HATEOAS, generando una aplicación autodescubrible.

Para un correcto diseño, debería alcanzarse al menos el nivel 2

**BUENAS PRÁCTICAS O CONSEJOS**

1. Usar SSL siempre. HTTPS, una conexión segura, encriptada.
2. Documentar la API con OpenAPI (Herramienta Swagger para Spring).
3. Versionar las API. No actualizar cambios sobre las anteriores, sino generar nuevas versiones con los cambios. Esto permite que la aplicación siga funcionando con versiones anteriores. La versión se indica en la URI, a continuación del servidor y del dominio.
4. Representar recursos (sustantivos) y no acciones ni formatos de representación. Tipos de recursos:
   1. Colecciones: Conjuntos de datos o de recursos. Por convención debe ir en plural.

/products

* 1. Documents: Una instancia de la colección. Por convención debe ir en singular

/prodcuts/1 (id)

/products/pencil (nombre)

* 1. Stores: Recursos que manejan el cliente

/users/20/favorites (lista de favoritos del usuario 20)

* 1. Controladores: Funciones ejecutables, acciones

/users/20/reset.password (Resetear password del usuario 20)

1. URIs: spinal-case (todo en minúscula separado por guiones simples)
2. Cuerpo: lowerCamelCase
3. Dentro del cuerpo de la respuesta puede haber errores que quedan disimulados, sin especificar. Mejor utilizar códgios HTTP para dar significado a la respuesta:
   1. 201: Recurso creado
   2. 202: Solicitud recibida. En proceso.
   3. 204: Solicitud exitosa. Respuesta sin contenido.
   4. 401: No autorizado
   5. 403: Acceso prohibido o denegado
   6. 404: Recurso no encontrado
   7. 405. Método no permitido
   8. 500: Error interno del servidor
4. No hacer todo con el método POST. Utilizar métodos HTTP para indica rla intención:
   1. GET: Obtener un recurso
   2. POST: Crear un nuevo recurso
   3. PUT: Actualizar un recurso existenteal completo
   4. DELETE: Eliminar un recurso
   5. PATCH: Actualiza el recurso especificando los campos a cambiar
   6. OPTIONS: Lo utilizan los navegadores para conectar y validar los ENDPOINTS.
5. Usar JSON en las respuestas, no cadenas de texto simples.
6. Ocultar las IDs de los recursos de la BD mediante código HASH

**DOCUMENTAR UNA API REST**

* Describir la acción que representa la petición y su verbo HTTP
* Indicar la URL del recurso sobre el que se efectúa la acción
* Como se invoca
* El formato en el que se enviará la información (JSON o XML)
* Cabeceras para indicar cómo se debe devolver la información
* Parámetros QUERY con criterios de filtrado
* Cuáles son las posibles respuestas
* Explica cada tipo de dato
* Códigos de error
* Una petición de ejemplo